Encoding Linear Logic with Interaction Combinators

Ian Mackie
CNRS (UMR 7650), LIX, École Polytechnique
91128 Palaiseau Cedex, France
E-mail: mackie@lix.polytechnique.fr

and

Jorge Sousa Pinto
Departamento de Informática, Universidade do Minho
Campus de Gualtar, 4710-057 Braga, Portugal
E-mail: jsp@di.uminho.pt

The purpose of this paper is to demonstrate how Lafont's interaction combinators, a system of three symbols and six interaction rules, can be used to encode linear logic. Specifically, we give a translation of the multiplicative, exponential and additive fragments of linear logic together with a strategy for cut-elimination which can be faithfully simulated. Finally, we show briefly how this encoding can be used for evaluating λ-terms. In addition to offering a very simple, perhaps the simplest, system of rewriting for linear logic and the λ-calculus, the interaction net implementation that we present has been shown by experimental testing to offer a good level of sharing, in terms of the number of cut-elimination steps (resp. β-reduction steps). In particular it performs better than all extant finite systems of interaction nets.

Key Words: Interaction nets, linear logic, cut-elimination, λ-calculus

1. INTRODUCTION

Interaction nets [8, 9] have become one of the standard tools for the study of *local* computation, in particular when *sharing* is of greatest importance. To cite one of the most well-known systems of interaction nets, Gonthier, Abadi and Lévy presented a system which captures both optimal sharing in linear logic [7] and the λ-calculus [6] (which is based on Lamping's algorithm [11]). This system has been very well studied, and its importance widely accepted.

Based on a very restricted form of graph rewriting, interaction nets are amongst the very few implementation techniques which capture explicitly *all* of the elements of computation,

including garbage collection and copying. With the absence of external machinery to capture these main computational tasks, interaction nets offer a more reliable and a more precise explication of the cost of a computation, by counting the number of graph rewriting steps. Each such step is a constant time operation, and they are all that there is to a computation. This is in sharp contrast to some other reduction systems, for instance β-reduction $(\lambda x.t)u \rightarrow_\beta t[u/x]$ (defined in terms of substitution) which is never a known constant time operation. Similar comments apply to the $\mathbf{S}$, $\mathbf{K}$ combinators, and term (or graph) rewriting systems in general.

The most important property of a system of interaction nets is that reduction is *local*, and guaranteed by construction to be strongly confluent: reduction steps commute with each other. This information tells us that each (constant time) reduction step may not duplicate nor erase other redexes, and we thus have a handle on the concept of sharing—a key element for any efficient implementation. The fact that each rewrite of a net takes place in its own space explains why external machinery to copy or erase nets is absent from the system. We remark that this kind of rewriting is very well suited for parallel execution, which has been investigated by the second author [16, 17].

Proof nets, which are the traditional syntax for linear logic, possess many similarities with interaction nets. The multiplicative fragment of proof nets is in fact an interaction net system (indeed, this is the very origin of interaction nets). However, proof nets still have *global* reduction steps for the exponentials: contraction, weakening, dereliction and the commutation rule. The challenge therefore for capturing linear logic in interaction nets is to encode the exponential cut-elimination steps in a local way. To achieve this we need a representation of proofs allowing the rewrite steps to be decomposed and implemented incrementally. For instance, weakening should gradually erase a proof and contraction should gradually copy a proof. Since there is no way that an active pair (the interaction net analogue of a redex) can be duplicated, this forces sharing in a very natural way.

Lafont's interaction combinators [10] are a fixed system of interaction nets which consists of three agents and six interaction rules. Lafont demonstrated that this extremely simple system of rewriting is *universal*—any other system of interaction nets can be simulated in it (we also note that interaction nets are Turing complete: they can simulate a Turing machine). This important result in interaction nets is analogous to the functional completeness of $\mathbf{S}$ and $\mathbf{K}$ in combinatory logic. The purpose of this paper is to study the possibility of using these interaction combinators for the encoding of cut-elimination in linear logic [5]. Using known translations of the λ-calculus into linear logic, we also obtain an encoding of β-reduction.

There are two alternative choices available to us for such an encoding:

1. If we take any existing system of interaction nets for linear logic (see below), then it is possible to apply the main result of interaction combinators ([10, Theorem 1]) to obtain an interaction combinator encoding of each agent of an existing system, which simulates the original system.

2. Alternatively, we can try to use the combinators in a more direct way by giving a natural translation of proofs in linear logic into interaction combinators. We note that many of the constructions used in the proof of the above mentioned theorem were inspired by linear logic in the first place.

The disadvantage with the first approach is that the encodings are very complicated, and moreover they just mimic reduction in the original system with a great deal more reduction

steps. Thus with this approach there is no possibility of discovering any new strategies for reduction, and little, if any, insight will be gained into the cut-elimination procedure. The second approach offers a greater challenge: in particular there is hope that new strategies of reduction in the underlying logic may be brought out. In this paper we show that this is indeed the case: by generalizing some of the results of interaction combinators, we obtain such a system in a very direct way.

Over the last ten years, three other systems of interaction nets have been developed to capture cut-elimination for (multiplicative exponential) linear logic, and thus also reduction in the λ-calculus:

- Gonthier, Abadi and Lévy gave the first system for linear logic and the λ-calculus, which encodes optimal reduction (no cut or redex will ever be duplicated), as defined by Lévy [12]. This particular system is defined using an infinite set of agents and rules, and captures a variant of linear logic (using the alternative functorial promotion rule).
- Abramsky defined a very simple finite system of interaction nets, which has been studied by the first author [13]. This system requires 8 agents and 16 interaction rules, and captures all of the cut-elimination steps of linear logic with the exception of the exponential commutation rule. In terms of the λ-calculus, this system offers a weak notion of reduction, but is nevertheless adequate for the evaluation of programs (reduction of closed terms to weak head normal form).
- More recently, a more complicated finite system of interaction has been developed which uses 13 agents and 38 interaction rules [15]. The novelty of this encoding is that nets in normal form correspond to the translation of cut-free proofs, without any restriction on the form of the conclusion.

Our aim in this paper is to provide an efficient implementation of cut-elimination in linear logic using local rewrite steps, using a *finite* system. It is clear why efficiency (when we count the number of interaction steps) is an important goal, but we also stress the importance of having a *finite* system of interaction nets. In fact, we can encode cut-elimination in linear logic with an infinite system, which will always generate a net representation of a cut-free proof with just one interaction: each proof is represented by an agent and an interaction with this agent will replace it with another agent representing the cut-free proof. We thus need an infinite set of agents to represent each proof in linear logic, and an infinite set of rules which perform cut-elimination. Of course, this example is exaggerated to make a point: we are interested in counting local rewrite steps to give an indication of the cost of a computation, and if additional (external) work is required, then this will obscure this measure.

With respect to other finite systems of interaction nets which encode linear logic, the system that we propose in this paper is by far the simplest one. Moreover, it is more efficient with respect to the number of cut-elimination steps performed—a good strategy for cut-elimination is therefore imposed. The prominent feature is that the exponential commutation rule (moving one box inside another in proof net terminology) is obtained *for free*: the translation of proofs into combinators is invariant under this commutation rule. One of the salient features of proof nets for linear logic is that they factor out all of the commutation rules of the sequent calculus, with the exception of this exponential commutation. The interaction net encoding that we give therefore can be seen as extending the proof net idea. It is also worth remarking that in terms of the λ-calculus, this can be understood as saying that substitutions can be pushed through abstractions for free, which

is a well-known problematic issue in the work on λ-calculus with explicit substitutions. It is thanks to this remarkable property that we achieve a good notion of sharing in this encoding.

However, as a consequence of the fine granularity of the combinators, the encoding does not offer the most efficient system with respect to the overall number of interactions. Nevertheless, we show benchmark results which justify that this system performs better than two of the extant systems.

Related work. This work is founded on understanding cut-elimination and reduction in the λ-calculus by using interaction nets. The way that interaction nets capture sharing in a natural way makes their use an interesting and novel approach to the implementation of languages based on β-reduction. The origins of this approach come from the work of Gonthier, Abadi and Lévy [6, 7], which is founded on Lamping's algorithm. Our aim is to find alternative systems of interaction nets, which offer alternative sharing strategies other than optimal reduction, but which are nevertheless efficient and simple. In contrast to this approach is the work of Asperti et al. (BOHM [1]) where the leading theme is the efficient implementation of optimal reduction.

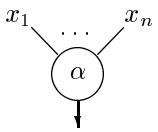
Many of the constructions used in this paper were inspired by, and rely heavily on, the work of Lafont [10]. Our contribution is to apply that work to give encodings of linear logic and the λ-calculus.

Overview. The rest of this paper is organized as follows. In the following section we recall interaction nets, specifically the system of interaction combinators and a number of constructions that we shall use throughout the paper. In Section 3 we give the encoding of proofs in linear logic into interaction combinators. In Section 4 we set up a strategy for cut-elimination in linear logic. Section 5 is devoted to showing how the encoding of linear logic imposes this strategy just using the interaction rules for the combinators. Section 6 shows how the results can be extended to cover the additives of linear logic. In Section 7 we briefly sketch how the λ-calculus can be implemented, and give some experimental results comparing this system of interaction with others mentioned in the introduction. Finally, we conclude the paper in Section 8.

Acknowledgement. Research partially supported by PRAXIS XXI grant BD/11261/97 and by EU TMR LINEAR "Linear Logic in Computer Science".

2. INTERACTION NETS

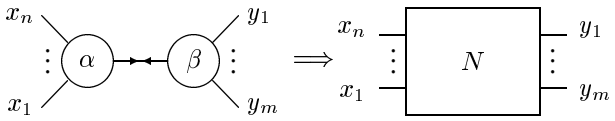
An interaction net system is specified by giving a set Σ of symbols, and a set $\mathcal{R}$ of interaction rules. Each symbol $\alpha \in \Sigma$ has an associated (fixed) *arity*. An occurrence of a symbol $\alpha \in \Sigma$ will be called an *agent*, which we draw as:



If the arity of α is n, then the agent has $n+1$ *ports*: a distinguished one called the *principal port* depicted by an arrow, and n *auxiliary ports* labeled $x_1, \ldots, x_n$ corresponding to the arity of the symbol. We index ports clockwise from the principal port, hence the orientation of an agent is not important.

A net N built on Σ is a graph (not necessarily connected) with agents at the vertices. The edges of the graph connect agents together at the ports such that there is at most one edge at every port (edges may connect two ports of the same agent). The ports that are not connected to other ports are called the free ports of the net. There are two special instances of a net: a wiring (no agents), and the empty net.

A pair of agents $(\alpha, \beta) \in \Sigma \times \Sigma$ connected together on their principal ports is called an *active pair*; this is the interaction net analog of a redex. An interaction rule $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ replaces an occurrence of the active pair (α, β) by a net N. Rules have to satisfy a very strong condition that all the free ports are preserved during reduction, and moreover that there is at most one rule for each pair of agents. The following diagram illustrates the idea, where N is any net built from Σ.



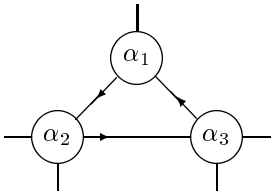
If a net does not contain any active pairs then we say that the net is in normal form. We use the notation $\Longrightarrow$ for a one step reduction and $\Longrightarrow^*$ for the transitive and reflexive closure. Additionally, we write $N \Downarrow N'$ if there is a sequence of interaction steps $N \Longrightarrow^* N'$, such that N' is a net in normal form.

As a direct consequence of the definition of interaction nets, in particular of the constraints on the interaction rules, reduction is strongly confluent [8]; indeed all reduction sequences are equivalent up to permutation. Consequently, we have the following additional properties of interaction nets:

LEMMA 2.1. *Let N be a net in an interaction system $(\Sigma, \mathcal{R})$, then:*

1.If $N \Downarrow N'$ then all reduction sequences are terminating (N is strongly normalizing).
2.Normal forms are unique: if $N \Downarrow N'$ and $N \Downarrow N''$ then $N' = N''$.

There is one interesting phenomenon that may arise in a net, which is called a *deadlock*. A net containing a cyclic path following principal ports is called a deadlocked net (also known as a vicious circle, or simply a cycle). The following configuration gives an example, where no interactions are possible, and in particular the net cannot be erased or duplicated:



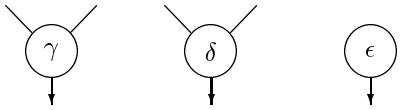
We note that such nets can also be created during reduction. All of the nets that we consider in this paper will be deadlock free.

2.1. Interaction Combinators

The first system of interaction combinators was presented by Gay [4], who defined a complete system of interaction using eight agents. With an ingenious encoding of a net

that can be duplicated, Lafont [10] managed to define a system using just three agents and six interaction rules. It is this latter system that we shall use throughout this paper.

The three agents of Lafont's system are: γ (a constructor of arity 2), δ (a duplicator of arity 2) and ϵ (an eraser of arity 0), which we draw in the following way:



Every net that we construct will be built by connecting occurrences of these agents together. In Figure 1 we give the six interaction rules for this system. The aim of this paper is to show how this system of combinators can be used to encode both cut-elimination in linear logic and β-reduction in the λ-calculus.

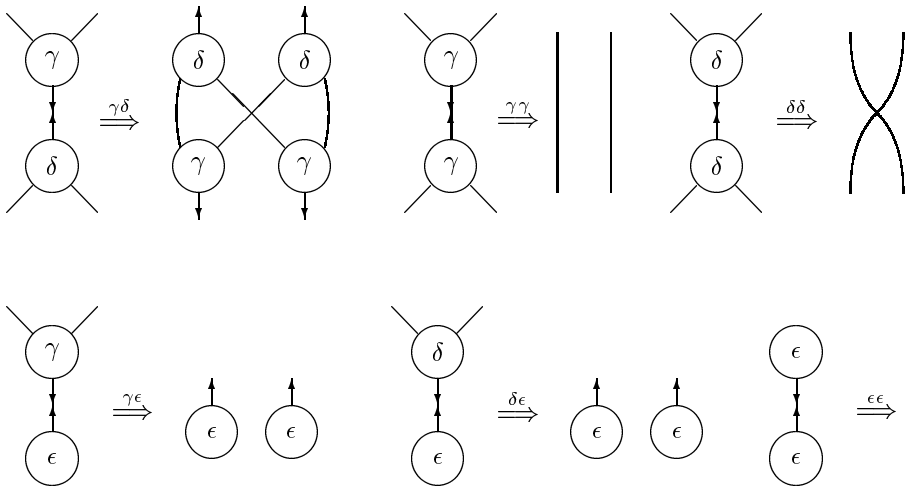
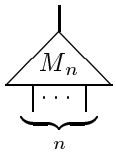


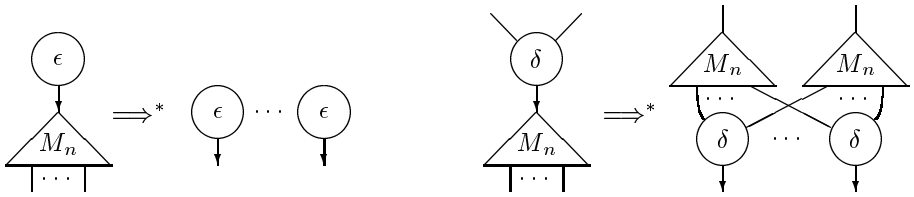
FIG. 1. Interaction Rules

Multiplexing. It will be useful for the translation, and for reasoning about nets, to provide some abbreviations (macros) which are built out of the combinators. The constructor agent γ can be used as a binary multiplexing agent: it groups the two edges on the auxiliary ports into one edge on the principal port. Symmetrically, γ can also be used as a demultiplexing agent, and the $\gamma\gamma$ interaction rule can then be understood as removing the shared edge between the agents. This idea can be generalized to n-ary multiplexing nets as we now explain.

DEFINITION 2.1. A net M_n, which groups n edges into one, drawn in the following way:

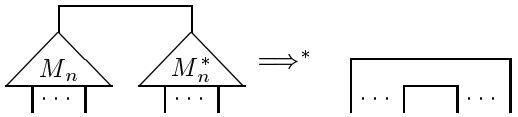


is a *multiplexing net* if it can be erased with ϵ and duplicated with δ:



Note that for these two properties to hold the net M_n must be free from δ agents, and also free from active pairs and deadlocks.

DEFINITION 2.2. A pair of nets (M_n, M_n^*) is a *multiplexing pair* when both M_n and M_n^* are multiplexing nets which also satisfy the following property:



For this pair of nets, we shall say that M_n^* is the demultiplexing net of M_n.

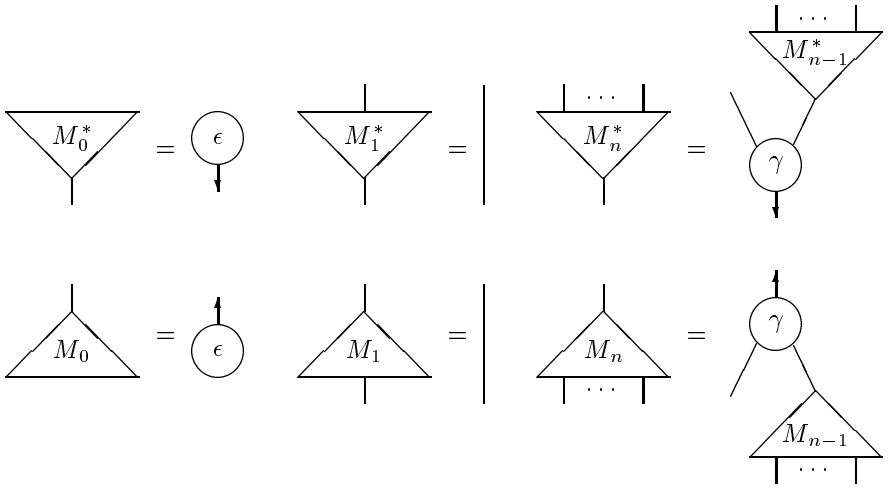


FIG. 2. Constructing Multiplexing Nets

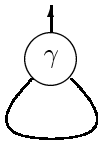
LEMMA 2.2.

1.Multiplexing nets: For all $n \geq 0$, M_n can be built without using δ or ϵ.

2.Multiplexing pairs: For all $n \geq 1$, (M_n, M_n^*) can be built without using δ or ϵ. For $n = 0$, (M_n, M_n^*) can be built without using δ.

Proof.

1. Define M_0 as the following net:

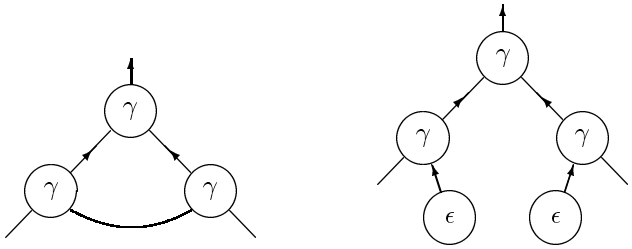


and for M_n, $n > 0$, see the lower half of Figure 2.

2. One possible way is given in Figure 2, which shows the duality between M_n and M_n^*. Note that if we try to build M_0 without ϵ (as in the case above) then we cannot find an M_0^* without ϵ.

In both cases, it is easily seen that the respective properties of multiplexing nets and multiplexing pairs are satisfied. ■

The above lemma is crucial for the encodings that we give later, as we need a way of building nets which are free from δ and ϵ. Of course, other possibilities exist for building multiplexing nets and pairs, for instance (M_2, M_2^*) can be built in the following way:



For the rest of this paper *multiplexing nets* will be built using only γ agents, and *multiplexing pairs* will be built from γ and ϵ. However, whenever we use a specific (fixed, known size) multiplexing pair (M_n, M_n^*), $n > 1$, in a definition, we shall always assume that it is constructed out of γ agents alone, and thus pairs like (M_2, M_2^*) above will not be considered.

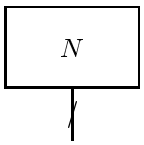
Note that as a consequence of the properties of the multiplexing nets, we can have a universal system of interaction defined from M_n ($n > 1$), δ and ϵ. Thus the net M_n can be treated as an agent of arity n. Indeed, we shall see that our translations of linear logic and the λ-calculus use the nets M_5 and M_6 very heavily, and we have found that it offers a reasonable improvement to include these nets as agents in the system.

2.2. Packing Nets

The main use that we make of multiplexing nets is for constructing packages, which for this paper are nets that do not contain either δ or ϵ agents. This concept is an extension of the notion of a package used in [10], and will be used later for the encoding of the promotion rule in linear logic. The process of building a net entirely out of γ agents is a two phase process:

1. first we extract the δ agents,
2. then we extract the ϵ agents, *including those possibly introduced in the first step*.

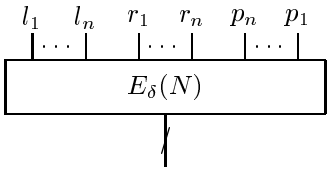
We now explain in some detail how this is achieved. We begin with the δ extraction, and consider a net N built using the interaction combinators, which has a number of free edges:



where the bus notation is used to keep the diagrams simple. Erasing a δ agent from this net creates three additional free edges, corresponding to the left auxiliary port (l), right auxiliary port (r) and the principal port (p) of the erased agent, where we use the following convention for the orientation of the edges:

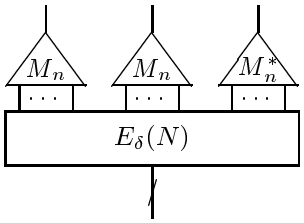
Suppose there are n occurrences of δ in the net N, indexed $\delta_1, \ldots, \delta_n$. For each δ_i we associate the triple (l_i, r_i, p_i), which are the corresponding free edges created when δ_i is removed.

We define the net $E_\delta(N)$ to be N where all occurrences of δ have been removed, and the $3n$ additional free edges grouped in the following way:



Remark that the order of the p_i edges is reversed with respect to that of l_i and r_i. Next, let (M_n, M_n^*) be a multiplexing pair of arity n. We then group the l_i edges together using M_n to give a single edge. The same can be done for the r_i and p_i edges, except that the p edges will be grouped using M_n^* rather than M_n (this is important for the dynamics of the system):

$P_\delta(N) \stackrel{\text{def}}{=}$



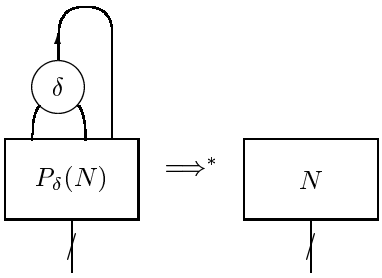
By Lemma 2.2, part 2, the multiplexing pair (M_n, M_n^*) can be built without using δ agents, and thus we have obtained a net free from δ agents, with three additional edges. We call this resulting net $P_\delta(N)$, which is the definition of the δ *package of* N. Note that if there are no δ agents in N, then we use M_0 and M_0^*, which can be ϵ agents, to construct the additional three free edges. We emphasize that $P_\delta(N)$ may contain *more* ϵ agents than N.

There are two factors that make this packing non-unique:

1. the indexing on the δ agents has been done in an arbitrary way,
2. the construction of the multiplexing pair (M_n, M_n^*) is left unspecified.

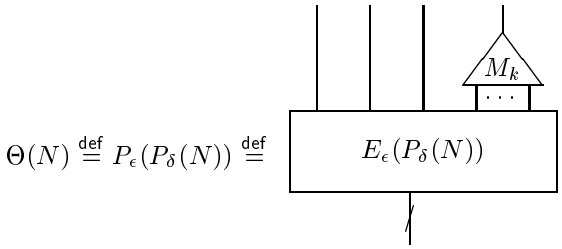
The following lemma shows that this is not important, as the contents of a packaged net can always be recovered be re-introducing the δ agents.

LEMMA 2.3. *Let N be a net built out of γ, δ, ϵ. For any indexing on the δ agents in N, and any multiplexing pair (M_n, M_n^*), the following reduction sequence is possible:*



Proof. The net M_n^* can be duplicated with the δ agent, and then both occurrences of the multiplexing pair (M_n, M_n^*) will annihilate, leaving n occurrences of δ connected to the correct edges. At no point in this proof is the construction of the multiplexing pair or the order of the edges used. ■

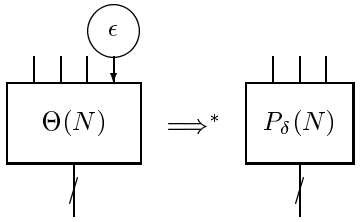
We next show how ϵ agents can be extracted from $P_\delta(N)$. The general idea follows the same pattern as for the δ agents. We define $E_\epsilon(P_\delta(N))$ to be the net $P_\delta(N)$ with all ϵ agents extracted. Assume that there are k ϵ agents in the net. If we erase them, then k free edges will be created, which can be combined together with a multiplexing net M_k. This gives us the following net, which now has one additional free edge:



By Lemma 2.2, part 1, M_k can be constructed uniquely out of γ agents, and thus the resulting net $P_\epsilon(P_\delta(N))$, which defines the ϵ *package of* $P_\delta(N)$, can be built entirely out of γ agents as required. It is important to note that if there are no ϵ agents in $P_\delta(N)$, then we use M_0 constructed without ϵ, as shown previously. As with the case for P_δ, P_ϵ is not unique (for the same reasons). We write $\Theta(N)$ as an abbreviation for $P_\epsilon(P_\delta(N))$.

Analogous to Lemma 2.3, we can always recover $P_\delta(N)$ from $\Theta(N)$:

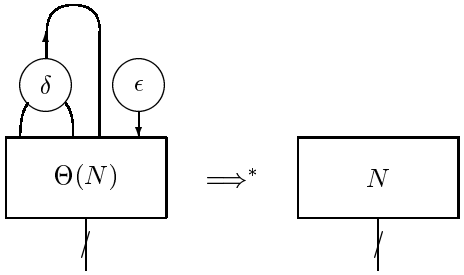
LEMMA 2.4. *Let N be a net built out of γ, δ, ϵ. For any indexing on the ϵ agents, and any multiplexing net M_k, the following reduction sequence is possible:*



Proof. M_k will be erased by the ϵ agent introduced, leaving k ϵ agents connected to the correct edges. If $k = 0$, the ϵ will erase the multiplexing net M_0. ■

We can now put Lemmas 2.3 and 2.4 to use to obtain the following:

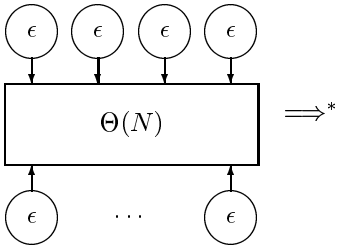
LEMMA 2.5 (Unpacking). *For any net N, N can be recovered from $\Theta(N)$ in the following way:*



We note that the net N is not necessarily in normal form, even when $\Theta(N)$ is.

Proof. By Lemma 2.4 we can recover the net $P_\delta(N)$, and then by Lemma 2.3 we can recover N. ■

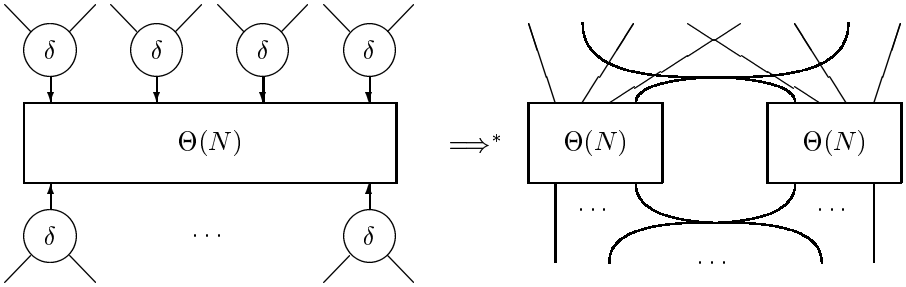
LEMMA 2.6 (Erasing). *If $\Theta(N)$ is a net in normal form (without deadlocks) then $\Theta(N)$ can be erased in the following way:*



where the right-hand side is the empty net.

Proof. This is a specific instance of a more general result that *any* net in normal form and without deadlocks can be erased. The proof is by induction on the size of the net to be erased. ■

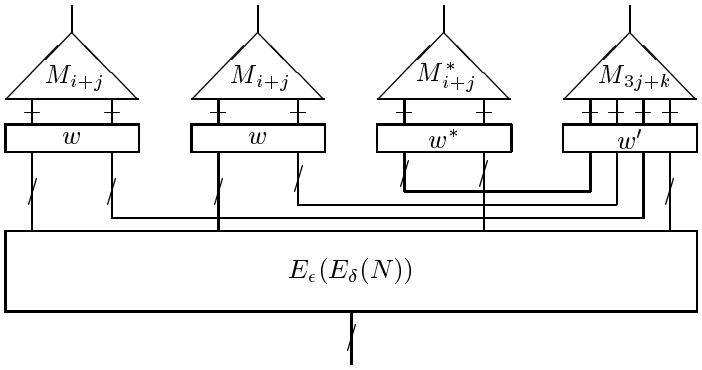
LEMMA 2.7 (Duplication). *If $\Theta(N)$ is a net in normal form (without deadlocks) then it can be duplicated in the following way:*



Proof. The proof follows a similar reasoning to the erasing lemma above. ■

Next we study the general form of the net $\Theta(N)$. As pointed out, the net $P_\delta(N)$ may introduce additional ϵ agents for the construction of the multiplexing pair. These additional agents are then extracted in the construction of $P_\epsilon(P_\delta(N))$. The following Lemma shows how we can understand $\Theta(N)$ as the extraction of δ and ϵ from N, together with a net containing multiplexing nets, which gives an alternative way of constructing packages.

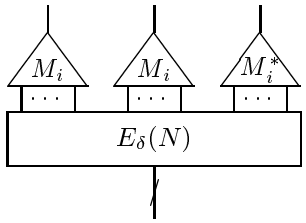
LEMMA 2.8 (Decomposition). *For any net N built out of γ, δ, ϵ, the net $\Theta(N)$ can be decomposed, according to the diagram below, as a net $E_\epsilon(E_\delta(N))$, which we call the kernel, and four multiplexing nets, which we call the package interface of $\Theta(N)$.*



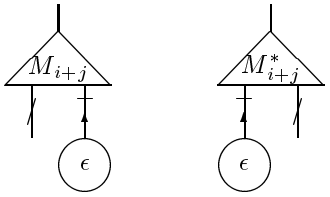
In the diagram, (M_{i+j}, M^*_{i+j}) is a multiplexing pair (built by P_δ), M_{3j+k} is a multiplexing net (built by P_ϵ), and $E_\epsilon(E_\delta(N))$ is the net obtained by extracting all the δ and ϵ agents from N. The nets w and w^* are symmetric wirings, and w' is a wiring.

Proof. To keep the proof simple, we assume without loss of generality, that the wiring nets are straight connections, and thus the multiplexing nets of the form M_{i+j} group two bundles of wires containing $i + j$ edges together. The net $P_\delta(N)$ has the following form, where we assume that N contains i occurrences of δ:

$P_\delta(N) =$



Assume that N contains k occurrences of ϵ. Since (M_i, M_i^*) is a multiplexing pair, M_i and M_i^* may also contain ϵ agents. Let j be the number of occurrences of ϵ in M_i (resp. M_i^*). Assume, without loss of generality, that M_i and M_i^* are constructed as follows, where M_{i+j} and M_{i+j}^* do not contain any occurrences of ϵ:

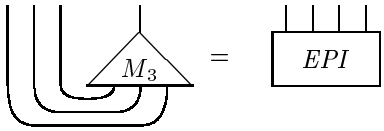


(If they are not of this form, then with an appropriate wiring we can move all the ϵ agents to one side.) Therefore P_ϵ will extract the k occurrences of ϵ from N, and j occurrences of ϵ from each of the three multiplexing nets, which gives $3j + k$ additional free edges which can be collected together with a multiplexing net M_{3j+k}.

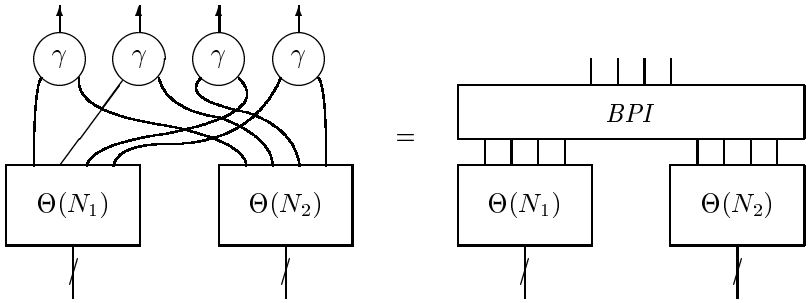
This justifies the package interface of $\Theta(N)$, and it is straightforward to see that the kernel is $E_\epsilon(E_\delta(N))$. ■

This proof shows that $\Theta(N)$ can be constructed in different ways, and as we shall show below, these differences are not important to us. As an aside, we remark that $E_\epsilon(E_\delta(N)) = E_\delta(E_\epsilon(N))$, but note that $P_\epsilon(P_\delta(N)) \neq P_\delta(P_\epsilon(N))$ because ϵ agents may be introduced by P_δ.

It is worth pointing out two specific instances of the package interface of $\Theta(N)$. First, if N does not contain any occurrences of ϵ or δ, ($i = 0$, $k = 0$), then the interface is the following net, to which we give the name EPI (Empty Package Interface).



Note that the net EPI can be built using only γ agents. Next, if we want to combine two packaged nets, then this can be done in the following way, where the diagram on the left shows one way of building the package interface:



Since this situation arises many times in our encodings, we will use the abbreviation BPI (Binary Package Interface) as shown in the diagram above. The net BPI does not contain either δ or ϵ, and thus the resulting net is indeed a package. It is straightforward to see that unpacking this combined package will give the nets N_1 and N_2 side by side.

The decomposition property allows us to define a notion of equivalence of nets, which abstracts away from the details of how the multiplexing pairs are built.

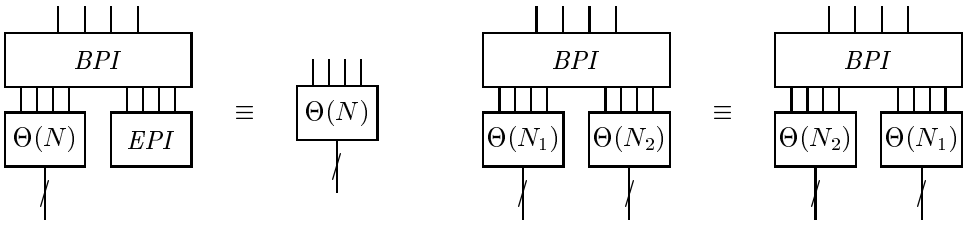
DEFINITION 2.3. Two packaged nets N and N' are said to be equivalent $N \equiv N'$ iff

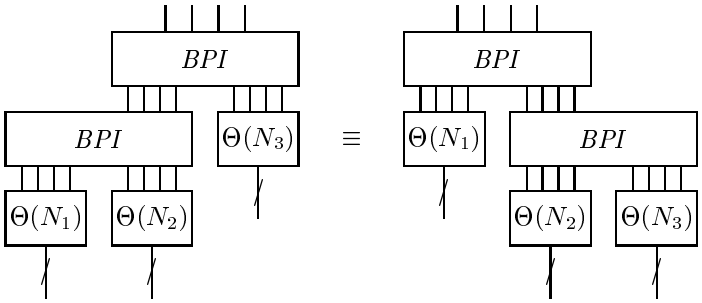
- they have the same package interface modulo the construction of the multiplexing pair M_{i+j}, M^*_{i+j}, the multiplexing net M_{3j+k}, and j, and
- the kernels are identical, or if there are nested packages, these packages are equivalent.

Because any two packaged nets which differ only by the package interface are equivalent, and moreover the unpacking of nets does not take into account how the package interface was constructed, then there is an alternative way of understanding the above equivalence. Two packaged nets N and N' are equivalent iff they both unpack to identical nets (if there are no nested packages), or if there are nested packages, these packages are equivalent.

Equivalence of nets will play a crucial role in the encoding of linear logic, because during reduction package interfaces will be created in arbitrary ways. The following lemma factors out the main uses of equivalences that we shall use, which also shows some properties of the BPI and EPI nets.

LEMMA 2.9. *The following three pairs of nets are equivalent, which shows that EPI is a unit for BPI, and BPI is both commutative and associative.*





Proof. We show that each pair of nets has the same decomposition. We detail the first case. By Lemma 2.8, $\Theta(N)$ has a package interface built from M_{i+j}, M_{i+j}, M^*_{i+j}, and M_{3j+k}. Expanding the nets BPI and EPI gives a decomposition of the left-hand side as M_{i+j+1}, M_{i+j+1}, M^*_{i+j+1}, and $M_{3(j+1)+k}$, which is the same decomposition as $\Theta(N)$ modulo j. The other two cases follow in the same way, where the only difference in the decompositions is the way in which the multiplexing pairs and nets are constructed. ■

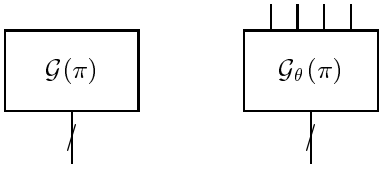
It is straightforward to see that in each case, each pair of nets unpacks to the same net.

This completes the general properties of the combinators that we use for the rest of the paper. We next give the actual encodings of linear logic proofs.

3. ENCODING LINEAR LOGIC

In this section we give an encoding (which we can see as a compilation) of linear logic proofs into interaction combinators—every proof will be represented as a net built up from the agents γ, δ and ϵ. We define simultaneously two translations: the first one, $\mathcal{G}(\pi)$, is the actual representation of a proof π, and a second, $\mathcal{G}_\theta(\pi)$, is the corresponding package that will be used as an auxiliary construction for $\mathcal{G}(\pi)$. In fact $\mathcal{G}_\theta(\pi)$ is just an abbreviation for $\Theta(\mathcal{G}(\pi))$, and thus, as shown in the last section, the difference between the two is that $\mathcal{G}_\theta(\pi)$ does not contain any occurrences of the agents δ or ϵ (it is built entirely from γs).

Let π be a proof with conclusion $\Gamma = A_1, \ldots, A_n$. The two translations will have the following general form, where we use the bus notation to keep the diagrams simple:



The n free edges at the bottom correspond to Γ (one edge for each formula, in the correct order to avoid labeling the free edges). The net $\mathcal{G}_\theta(\pi)$ has four additional free edges at the top, which correspond to the free edges created by extracting the δ and ϵ agents.

The purpose of a package $\mathcal{G}_\theta(\pi)$ in the translation is to encode the promotion rule of linear logic. Such a net must have the possibility to be duplicated, erased and opened (to recover the net $\mathcal{G}(\pi)$) which correspond to the contraction, weakening and dereliction cut-elimination steps respectively. It will be essential that the net $\mathcal{G}_\theta(\pi)$ does not contain any occurrences of the agent δ so that it can be successfully duplicated by a δ agent. It is not strictly essential that the package is free from ϵ agents, however it facilitates some

of the proofs of correctness given later. It also provides a novel extension to the idea of a package (due to Lafont) which allows proofs to be encoded using only the γ agent.

We now give the inductive definitions of the translation of proofs into interaction combinators, showing side-by-side $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$.

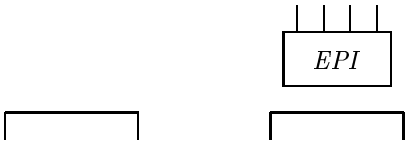
3.1. Identity Group

The identity group of linear logic consists of the axiom and the cut rule:

$$\frac{}{A^\perp, A}\,(\mathsf{Ax}) \qquad\qquad \frac{\Gamma, \boldsymbol{A} \qquad \boldsymbol{A^\perp}, \Delta}{\Gamma, \Delta}\,(\mathsf{Cut})$$

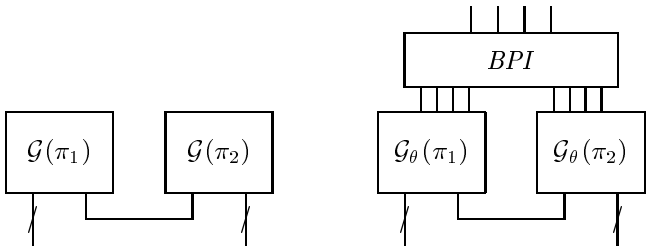
where, as usual, A need not be an atomic formula.

- If π is an axiom, then $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are given respectively as:



Note that there are no occurrences of δ or ϵ in the representation of the axiom, and thus $\mathcal{G}_\theta(\pi)$ uses the net EPI to build the correct interface. The net $\mathcal{G}_\theta(\pi)$ does not contain either δ or ϵ (we have already assumed that the net M_3 used to construct EPI does not contain either δ or ϵ) and has the correct interface, and thus satisfies the requirements for the $\mathcal{G}_\theta(\cdot)$ translation.

- Let π_1 be a proof of Γ, A and π_2 be a proof of $A^\perp, \Delta$. A proof π of Γ, Δ can then be built using the cut rule, and the nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined respectively as:



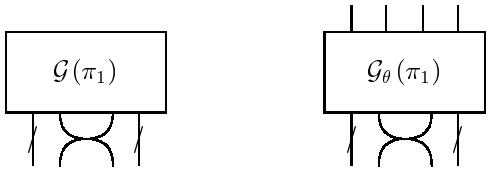
The net $\mathcal{G}(\pi)$ adds an edge connecting the representations of the conclusions A and $A^\perp$ from $\mathcal{G}(\pi_1)$ and $\mathcal{G}(\pi_2)$ respectively. The additional structure required to construct $\mathcal{G}_\theta(\pi)$ is simply the BPI net.

3.2. Structural Group

The structural group of linear logic consists of just the exchange (X) rule, which allows elements of a sequent to be permuted:

$$\frac{\Gamma, A, B, \Delta}{\Gamma, B, A, \Delta}\,(X)$$

This is reflected in the translation by simply exchanging the interface of the net by crossing over two free edges. If π_1 is a proof of Γ, A, B, Δ, then a proof π can be built using the exchange rule. The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are built in the following way:



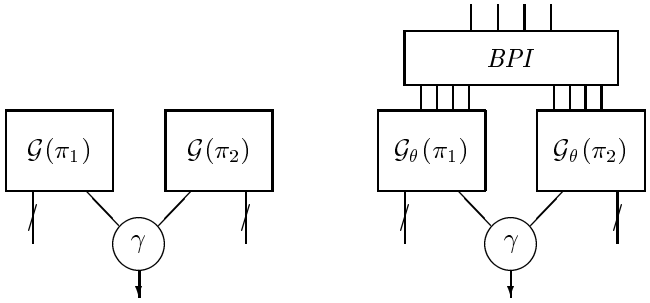
In particular, no agents are used, and the interface at the top of the net $\mathcal{G}_\theta(\pi)$ is the same as that for $\mathcal{G}_\theta(\pi_1)$.

3.3. Multiplicatives

The multiplicative group of linear logic consists of the tensor ($\otimes$) and the par ($\parr$) rules:

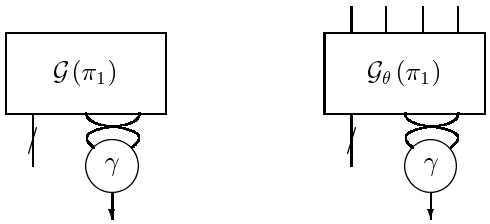
$$\frac{\Gamma, A \qquad B, \Delta}{\Gamma, A \otimes B, \Delta}\,(\otimes) \qquad\qquad \frac{\Gamma, A, B}{\Gamma, A \parr B}\,(\parr)$$

- Let π_1 be a proof of Γ, A and π_2 be a proof of B, Δ, then a proof π of $\Gamma, A \otimes B, \Delta$ can be built using the tensor rule ($\otimes$). The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined in the following way, where the agent γ is used to encode $\otimes$:



These constructions are very similar to the encoding of the cut rule discussed previously. The only difference is that the free edges representing the conclusions A and B are connected to the auxiliary ports of the γ agent, which has its principal port representing the conclusion $A \otimes B$.

- Let π_1 be a proof of Γ, A, B, then a proof π of $\Gamma, A \parr B$ can be built using the par rule ($\parr$). The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined respectively as the following, where $\parr$ is represented by the γ agent:



For both $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ the auxiliary ports of the γ agent represent the premises B and A in the rule, and the principal port represents the conclusion $A \parr B$. Note that the premises are twisted, which is necessary to capture the multiplicative cut-elimination step using the $\gamma\gamma$ interaction rule. Since there are no new δ or ϵ agents introduced in the translation, the top interface of the net $\mathcal{G}_\theta(\pi_1)$ is left unchanged.

We remark that we have used the γ agent to represent both $\otimes$ and $\parr$. Since we are translating sequent calculus proofs, cuts between two $\otimes$ (or two $\parr$) agents will never occur.

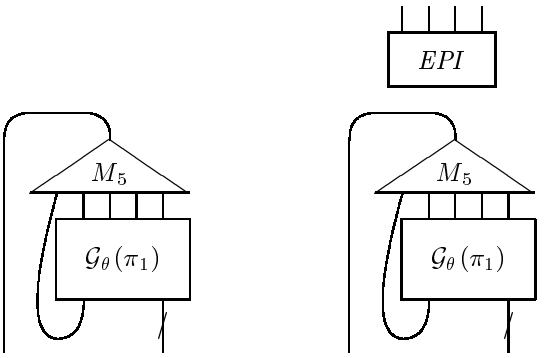
3.4. Exponentials

The exponential group of linear logic consists of the promotion $(!)$, dereliction (D), weakening (W), and contraction (C) rules:

$$\frac{A, ?\Gamma}{!A, ?\Gamma}\,(!) \qquad \frac{\Gamma, A}{\Gamma, ?A}\,(D) \qquad \frac{\Gamma}{\Gamma, ?A}\,(W) \qquad \frac{\Gamma, ?A, ?A}{\Gamma, ?A}\,(C)$$

The encoding of the exponential rules brings out some of the most interesting aspects of the translation. The delicate rule to encode is clearly the promotion rule, because the cut-elimination procedure will allow for promoted proofs to be copied (by the contraction cut-elimination step) and erased (by the weakening cut-elimination step). Clearly the agents δ and ϵ seem appropriate for this task, but the problem lies in the fact that a net containing a δ agent cannot be duplicated by the δ agent (cf. the interaction rule for two δ agents). However, the construction $\mathcal{G}_\theta(\pi)$ is precisely designed for this purpose, and thus will be used for the encoding of the promotion rule. Additionally, the dereliction cut-elimination step must allow for the net $\mathcal{G}(\pi)$ to be recovered from $\mathcal{G}_\theta(\pi)$. This will be achieved by re-introducing the extracted δ and ϵ agents, as we shall discuss later.

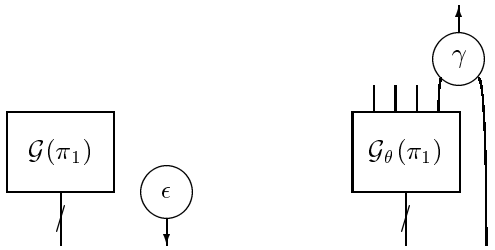
- Let π_1 be a proof of $A, ?\Gamma$, and π the proof of $!A, ?\Gamma$ built from π_1 using the promotion rule (!). The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined respectively as:



Observe that $\mathcal{G}(\pi)$ is defined in terms of $\mathcal{G}_\theta(\pi_1)$, which gives the encoding of boxes as packages. The net M_5, which is part of a multiplexing pair (M_5, M_5^*), groups together all of the free edges representing the extracted δ and ϵ agents, together with the conclusion A of the proof π_1. The top of the net M_5 represents the conclusion $!A$. Later we will see that the corresponding demultiplexing net M_5^* is used in the encoding of the dereliction, which will open the package $\mathcal{G}_\theta(\pi_1)$ under cut-elimination.

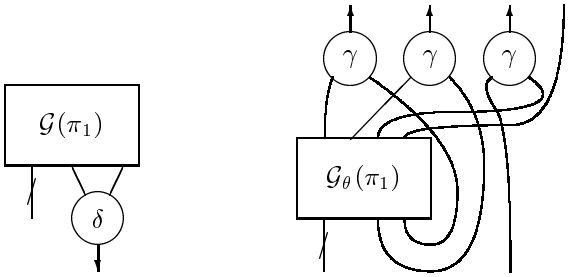
Since $\mathcal{G}(\pi)$ does not contain any occurrences of either δ or ϵ, we use the net EPI to obtain $\mathcal{G}_\theta(\pi)$ from $\mathcal{G}(\pi)$.

- Let π_1 be a proof of Γ, and π a proof of $\Gamma, ?A$ built from π_1 using the weakening rule (W). The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined respectively as the following, where an ϵ agent is used to represent weakening:



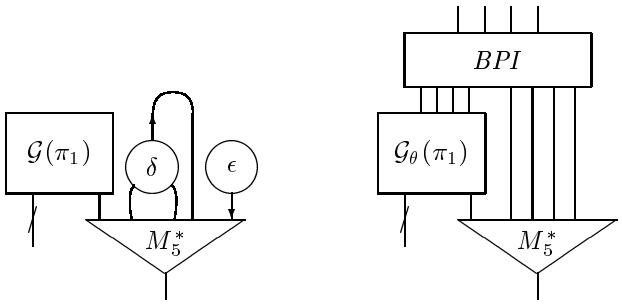
The principal port of the ϵ agent in $\mathcal{G}(\pi)$ corresponds to the conclusion $?A$. For the net $\mathcal{G}_\theta(\pi)$ we must extract this ϵ, using a γ agent as shown. The package interface of the net $\mathcal{G}_\theta(\pi)$ is thus the same as that of $\mathcal{G}_\theta(\pi_1)$, except that the multiplexing net M_k is extended to M_{k+1}. Remark that for the net $\mathcal{G}_\theta(\pi)$, there is no principal port corresponding to the conclusion $?A$. As we will see, this is reflected in the dynamics of the system: weakening cut-elimination steps will be forbidden inside packages.

• Let π_1 be a proof of $\Gamma, ?A, ?A$, and π the proof of $\Gamma, ?A$ built from π_1 using the contraction rule (C). The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined respectively as the following, where δ is used to represent contraction:



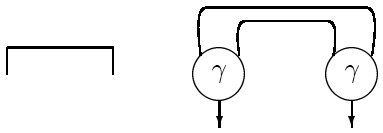
The auxiliary ports of the δ agent used in $\mathcal{G}(\pi)$ correspond to the two $?A$ premises of the rule, and the principal port corresponds to the conclusion $?A$. In the net $\mathcal{G}_\theta(\pi)$ this δ agent is extracted, and the free edges representing its left, right and principal ports are then multiplexed with those coming from $\mathcal{G}_\theta(\pi_1)$ using γ agents. The package interface of the net $\mathcal{G}_\theta(\pi)$ is thus the same as $\mathcal{G}_\theta(\pi_1)$, except that we have extended the multiplexing pair (M_n, M_n^*) to (M_{n+1}, M_{n+1}^*). Again, the free port of $\mathcal{G}_\theta(\pi)$ corresponding to the conclusion $?A$ of π is not connected to the principal port of an agent, and thus contraction cut-elimination steps will not be simulated inside packages.

• Let π_1 be a proof of Γ, A, and π the proof of $\Gamma, ?A$ built from π_1 using the dereliction rule (D). The nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are defined respectively as the following nets:



The encoding of dereliction uses the demultiplexing net M_5^*, corresponding to the multiplexing net M_5 used in the encoding of the promotion rule. For both $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$, the leftmost edge of the M_5^* net corresponds to the premise A in the rule, and the bottom edge of the net M_5^* corresponds to the conclusion $?A$. When this net is connected to the encoding of a promoted proof, the multiplexing pair (M_5, M_5^*) will interact resulting in the δ and ϵ agents being connected to the interface of the package that must be opened (i.e., a net $\mathcal{G}(\pi)$ will be recovered from $\mathcal{G}_\theta(\pi)$, as a consequence of Lemma 2.5). The construction $\mathcal{G}_\theta(\pi)$ extracts the δ and ϵ agents, and multiplexes the free edges, as described in the previous cases. Note that in this case, the conclusion $?A$ is represented by a principal port in the net $\mathcal{G}_\theta(\pi)$, which was not the case for both weakening and contraction.

We end this section with several remarks on the translation functions. First, the encoding of the axiom and cut rules does not rely on A being an atomic formula. Indeed, given two different sequent calculus proofs of $A^\perp \parr B^\perp, A \otimes B$, we construct two different nets:



where the first net is an axiom, and the second net is an axiom expanded into a par link and a tensor link. The same remark of course can be made for proof nets. The translation does not commit one to use only atomic or non-atomic axioms.

The second remark concerns the inductive definition of the construction of the net $\mathcal{G}_\theta(\pi)$. It is clear that the construction given step-by-step above will not necessarily give the simplest package interface. This is easy to see if we construct a proof out of a cut of two axioms: the package interface will be two occurrences of EPI and the net BPI, whereas it would also be possible to construct this package interface from just one EPI net. However, since we are working modulo the package interface construction, this does not cause any problems (cf. Lemma 2.9). Moreover, having an inductive definition of the net $\mathcal{G}_\theta(\pi)$ allows us to reason more directly about it, and to prove properties by induction.

4. A STRATEGY FOR CUT-ELIMINATION

In this section we define a strategy for cut-elimination in linear logic, specifically for the exponentials. Before defining our strategy, we begin by recalling in Figure 3 the general form of the cut-elimination steps for linear logic [5]. For a cut-elimination step to apply, the cut formula (highlighted in bold) must coincide with the principal formula of the premises of the rule. Otherwise, if the cut involves an auxiliary formula of one of the premises, then we can apply one of the commutation rules which are given in Figure 4. In this figure there is an additional case for $\otimes$ when C occurs in π_3 rather than π_2. The commutation rules, together with the fact that cut is associative, can be used to create a cut of the required form for the cut-elimination steps to apply. Cut-elimination steps can be applied anywhere in the proof, in particular above a promotion rule (within a box in proof net terminology). These rules are complete for cut-elimination: any proof can be transformed under these rules to give a cut-free proof, and moreover the process terminates.

We now impose a strategy on these rules, which is achieved by constraining the standard rules above, thus offering a weak form of cut-elimination. The resulting system will no longer be complete for cut-elimination: it will still be terminating, but will only terminate

$$\dfrac{\dfrac{\pi}{\Gamma, \boldsymbol{A}} \qquad \dfrac{}{\boldsymbol{A^\perp}, A}\,(\mathsf{Ax})}{\Gamma, A}\,(\mathsf{Cut}) \quad \overset{(AC)}{\Longrightarrow} \quad \dfrac{\pi}{\Gamma, A}$$

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, A} \qquad \dfrac{\pi_2}{B, \Gamma'}}{\Gamma, \boldsymbol{A \otimes B}, \Gamma'}\,(\otimes) \qquad \dfrac{\dfrac{\pi_3}{A^\perp, B^\perp, \Delta}}{\boldsymbol{A^\perp \parr B^\perp}, \Delta}\,(\parr)}{\Gamma, \Gamma', \Delta}\,(\mathsf{Cut}) \quad \overset{(\otimes,\parr)}{\Longrightarrow} \quad \dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{A}} \qquad \dfrac{\dfrac{\pi_2}{\boldsymbol{B}, \Gamma'} \qquad \dfrac{\pi_3}{A^\perp, \boldsymbol{B^\perp}, \Delta}}{\Gamma', \boldsymbol{A^\perp}, \Delta}\,(\mathsf{Cut})}{\Gamma, \Gamma', \Delta}\,(\mathsf{Cut})$$

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, A^\perp}}{\Gamma, \boldsymbol{?A^\perp}}\,(D) \qquad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{\Gamma, ?\Delta}\,(\mathsf{Cut}) \quad \overset{(!,D)}{\Longrightarrow} \quad \dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{A^\perp}} \qquad \dfrac{\pi_2}{\boldsymbol{A}, ?\Delta}}{\Gamma, ?\Delta}\,(\mathsf{Cut})$$

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma}}{\Gamma, \boldsymbol{?A^\perp}}\,(W) \qquad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{\Gamma, ?\Delta}\,(\mathsf{Cut}) \quad \overset{(!,W)}{\Longrightarrow} \quad \dfrac{\dfrac{\pi_1}{\Gamma}}{\Gamma, ?\Delta}\,(W)$$

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, ?A^\perp, ?A^\perp}\,(C)}{\Gamma, \boldsymbol{?A^\perp}} \qquad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{\Gamma, ?\Delta}\,(\mathsf{Cut}) \quad \overset{(!,C)}{\Longrightarrow} \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, ?A^\perp, \boldsymbol{?A^\perp}} \qquad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{\Gamma, ?\Delta, \boldsymbol{?A^\perp}}\,(\mathsf{Cut}) \qquad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{\dfrac{\Gamma, ?\Delta, ?\Delta}{\Gamma, ?\Delta}\,(C)}\,(\mathsf{Cut})$$

FIG. 3. Cut-elimination steps

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \quad \dfrac{\dfrac{\pi_2}{C, \Delta, A, B}}{\boldsymbol{C}, \Delta, A \parr B}\,(\parr)}{\Gamma, \Delta, A \parr B}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \quad \dfrac{\pi_2}{\boldsymbol{C}, \Delta, A, B}}{\Gamma, \Delta, A, B}\,(\mathsf{Cut})}{\Gamma, \Delta, A \parr B}\,(\parr)$$

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \quad \dfrac{\dfrac{\pi_2}{C, \Delta, A} \quad \dfrac{\pi_3}{\Theta, B}}{\boldsymbol{C}, \Delta, \Theta, A \otimes B}\,(\otimes)}{\Gamma, \Delta, \Theta, A \otimes B}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \quad \dfrac{\pi_2}{\boldsymbol{C}, \Delta, A}}{\Gamma, \Delta, A}\,(\mathsf{Cut}) \quad \dfrac{\pi_3}{\Theta, B}}{\Gamma, \Delta, \Theta, A \otimes B}\,(\otimes)$$

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{B^\perp}} \quad \dfrac{\dfrac{\pi_2}{B, \Delta, A}}{\boldsymbol{B}, \Delta, ?A}\,(D)}{\Gamma, \Delta, ?A}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{B^\perp}} \quad \dfrac{\pi_2}{\boldsymbol{B}, \Delta, A}}{\Gamma, \Delta, A}\,(\mathsf{Cut})}{\Gamma, \Delta, ?A}\,(D)$$

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{B^\perp}} \quad \dfrac{\dfrac{\pi_2}{B, \Delta}}{\boldsymbol{B}, \Delta, ?A}\,(W)}{\Gamma, \Delta, ?A}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{B^\perp}} \quad \dfrac{\pi_2}{\boldsymbol{B}, \Delta}}{\Gamma, \Delta}\,(\mathsf{Cut})}{\Gamma, \Delta, ?A}\,(W)$$

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \quad \dfrac{\dfrac{\pi_2}{C, \Delta, ?A, ?A}}{\boldsymbol{C}, \Delta, ?A}\,(C)}{\Gamma, \Delta, ?A}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \quad \dfrac{\pi_2}{\boldsymbol{C}, \Delta, ?A, ?A}}{\Gamma, \Delta, ?A, ?A}\,(\mathsf{Cut})}{\Gamma, \Delta, ?A}\,(C)$$

$$\dfrac{\dfrac{\dfrac{\pi_1}{B, ?\Gamma, ?A^\perp}}{!B, ?\Gamma, \boldsymbol{?A^\perp}}\,(!) \quad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{!B, ?\Gamma, ?\Delta}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\pi_1}{B, ?\Gamma, \boldsymbol{?A^\perp}} \quad \dfrac{\dfrac{\pi_2}{A, ?\Delta}}{\boldsymbol{!A}, ?\Delta}\,(!)}{B, ?\Gamma, ?\Delta}\,(\mathsf{Cut})}{!B, ?\Gamma, ?\Delta}\,(!)$$

FIG. 4. Commutation Rules

with cut-free proofs under certain conditions that we impose on the form of the conclusion of the proof. The restrictions imposed will be quite austere, but nevertheless the resulting system will be adequate for the evaluation of proofs of the usual data-types encoded in linear logic. It is also sufficient for programs obtained via translations of the λ-calculus into linear logic, which is one possible application of this work. We also remark that this strategy is related, but not identical, to the strategy used in [15] for an alternative encoding of linear logic into interaction nets. Our strategy will then be used in the following section to prove properties about the interaction combinators encoding of linear logic, where it will be shown that the system of interaction can simulate weak reduction, and moreover nets in normal form will correspond to the translation of (weak) cut-free proofs.

There are two important aspects that we want to bring out for our definition of a weak strategy for cut-elimination: that of *restricted internal reduction* inside an exponential box, and that of *closed reduction* (certain exponential cut-elimination steps can only take place when the context of the promotion rule is empty). The first of these constraints is in fact inspired by the encoding of proofs given in the previous section: the encoding $\mathcal{G}_\theta(\pi)$ (which corresponds to a net inside an exponential box) is free of both δ and ϵ agents, and thus both the contraction and weakening cut-elimination steps will not be possible as internal reductions.

Notation. Let π be a proof ending with (an application of) the promotion rule with conclusions $!A, ?\Gamma$, then we call $?\Gamma$ the *context* of (that application of) the rule. In order to define our strategy formally as a set of constraints on the general system, it will be convenient to refer to the following basic notions:

- Closed proof: a proof ending in the promotion rule with an empty context. In proof net terminology, this corresponds to an exponential box with no auxiliary doors.
- Closed exponential reduction: an exponential cut-elimination step involving a closed proof.
- Internal reduction: a cut-elimination step which takes place within an exponential box (above an application of the promotion rule).
- External reduction: a cut-elimination step which takes place outside the scope of any exponential box.

We can now give the weak strategy for cut-elimination in linear logic.

DEFINITION 4.1. We write $\pi \Longrightarrow_w \pi'$ if π' is obtained from π by a cut-elimination step in the following constrained system (the reader is referred to Figures 3 and 4 for the definition of the cut-elimination and commutation steps):

- The contraction $(!, C)$ and weakening $(!, W)$ cut-elimination steps are not permitted as internal reductions;
- Both contraction $(!, C)$ and weakening $(!, W)$ cut-elimination steps can only be performed when the proof π_2 ending in the promotion rule is a closed proof, and moreover π_2 is weak cut-free.

If no rule can be applied then we say that the proof is *weak-cut-free*.

Notation. We write $\pi \Downarrow_w \pi'$ iff there is a sequence of weak cut-elimination steps $\pi \Longrightarrow_w^* \pi'$, and π' is weak cut-free.

The following result shows that this restricted form of cut-elimination is sufficient to obtain cut-free proofs in certain cases:

THEOREM 4.1. *Let π be a proof of Γ, where Γ does not contain any exponentials ($?$ or $!$). If π is weak cut-free, then π is cut-free.*

Proof. Assume for a contradiction that π is weak-cut-free, but not cut-free. Let h be the height of one of the lowest cuts in the proof (with respect to the conclusion). Since the proof is weak cut-free, the only possible cases for this cut are the following:

1. A contraction cut of the form:

$$\dfrac{\dfrac{\Gamma, ?A, ?A}{\Gamma, \boldsymbol{?A}}\,(C) \qquad \dfrac{A, ?\Delta}{\boldsymbol{!A}, ?\Delta}\,(!)}{\Gamma, ?\Delta}\,(\mathsf{Cut})$$

where the context $?\Delta$ is not empty. Now the only way to erase the context $?\Delta$ from the conclusion is by the use of a cut rule of height $h' < h$ lower in the proof. But since there are no lower cuts, the context $?\Delta$ must be amongst the conclusions, and thus we are led to a contradiction.

2. There is a similar case for a weakening cut against a non-closed proof, which follows exactly the same reasoning as above.

3. The final case is when there is a cut (weakening or contraction) in the scope of an exponential box. The pattern is now standard: the exponential box, with conclusion $!A$, cannot be involved in a cut lower in the proof, and thus $!A$ must occur in the conclusion contradicting our assumption.

This completes all the possible cases. ■

Remark. Note that although the condition on the theorem (no exponentials in the conclusion) is quite a strong one, it does not mean that there are no exponentials inside the proof. Moreover, the condition is actually stronger than necessary, and it is easy to find examples of proofs that reduce under weak cut-elimination to cut-free proofs that do not satisfy any of the conditions of the theorem. However, this is a sufficient condition which is simple to express.

5. PROPERTIES

The purpose of this section is to prove that the weak strategy for cut-elimination introduced in the previous section can be faithfully simulated by our encoding of linear logic. We show that for each weak-cut-elimination step $\pi \Longrightarrow_w \pi'$, there exists a sequence of interaction rules which transform $\mathcal{G}(\pi)$ into $\mathcal{G}(\pi')$. Moreover, if π is a cut-free proof, then $\mathcal{G}(\pi)$ is a net in normal form. Therefore, if there are no exponentials in the conclusion of

the proof, then, using the strong confluence property of interaction nets, we obtain the main result of this section, that $\pi \Downarrow_w \pi' \iff \mathcal{G}(\pi) \Downarrow \mathcal{G}(\pi')$.

We begin with some basic properties of translated nets $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$, defined in Section 3.

LEMMA 5.1.

1.If π is cut-free, then $\mathcal{G}(\pi)$ is a net in normal form (without deadlocks).

2.If π is weak-cut-free, then $\mathcal{G}_\theta(\pi)$ is a net in normal form (without deadlocks).

Proof.

1. It is a straightforward observation from the definition of the translation functions that only the use of the cut rule can connect active pairs together.

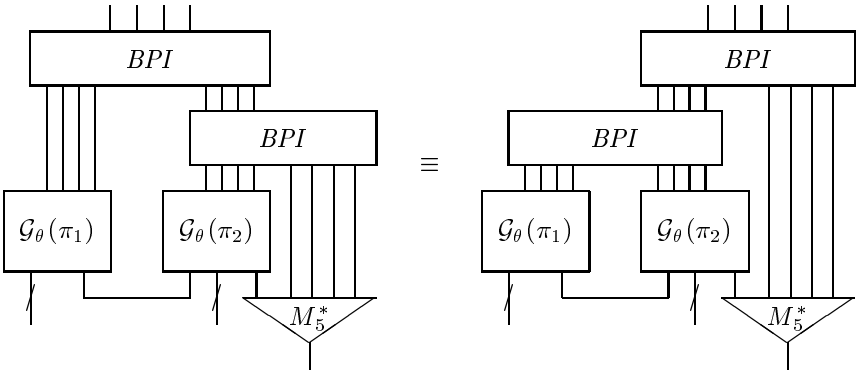
2. If π is cut-free, then the result follows by a similar observation as the previous case. If the proof is not cut-free, then there are two cases to consider: weakening and contraction cuts. In both cases the principal port of an exponential box is connected to the auxiliary port of a γ agent, and thus no active pair is created.

In both cases, the nets are deadlock free by inspection. ■

The encoding of sequent calculus proofs into interaction nets shares several properties with proof nets: the graphical representation factors out most of the sequential ordering of the rules imposed by the sequent calculus. The following result makes this precise.

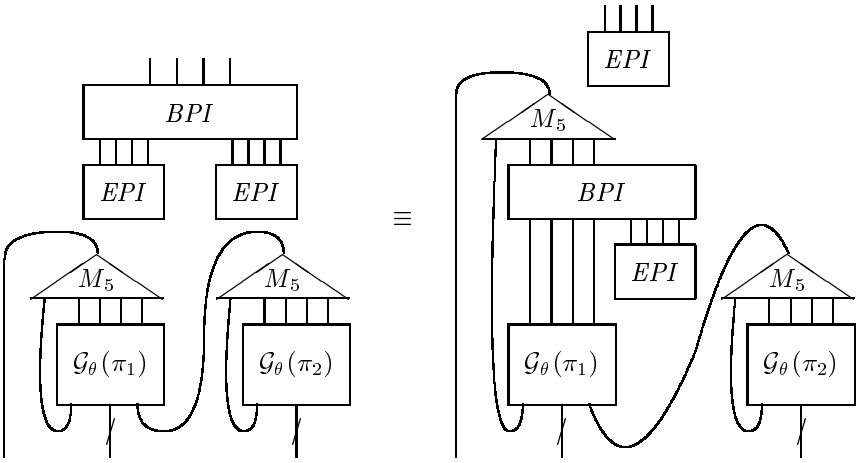
LEMMA 5.2. *If π and π' differ only by permutations of rules, or commutations of the cut rule (cf. Figure 4), then $\mathcal{G}(\pi) \equiv \mathcal{G}(\pi')$ and $\mathcal{G}_\theta(\pi) \equiv \mathcal{G}_\theta(\pi')$.*

Proof. For each case one draws the corresponding nets which are easily seen to be equivalent (cf. Definition 2.3). We use Lemma 2.9 to show the equivalence of the different ways in which the multiplexing nets are constructed. For instance, to show the dereliction commutation rule, we must show that the following two nets are equivalent:



which is the case by Lemma 2.9 (associativity of BPI nets).

To show the exponential commutation rule, the following nets must be equivalent:



For this we use twice the property that the net EPI is a unit for BPI, as given in Lemma 2.9. ■

Remark. Note that with respect to proof nets we have in addition the equivalence for the exponential commutation rule. This is in fact the main novelty and is one of the main motivations for studying this encoding of linear logic into interaction nets.

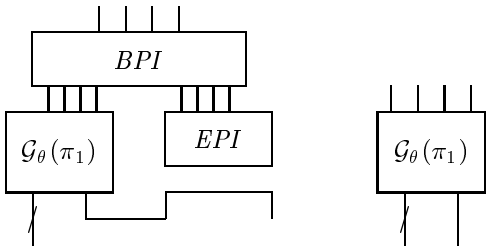
This completes the static properties for the translation. We now consider the dynamics of the system to show how weak cut-elimination is simulated.

LEMMA 5.3 (Package Reduction). *If $\pi \Longrightarrow_w \pi'$ by an internal reduction step, then there is a sequence of interactions such that $\mathcal{G}_\theta(\pi) \Longrightarrow^* N$, where $N \equiv \mathcal{G}_\theta(\pi')$.*

Proof. The proof proceeds by cases on the $\Longrightarrow_w$ relation. We draw the nets corresponding to both $\mathcal{G}_\theta(\pi)$ and $\mathcal{G}_\theta(\pi')$, and show that there is a sequence of interactions that can perform this transformation (modulo $\equiv$). Each case is straightforward, but relies very heavily on the Decomposition Lemma 2.8.

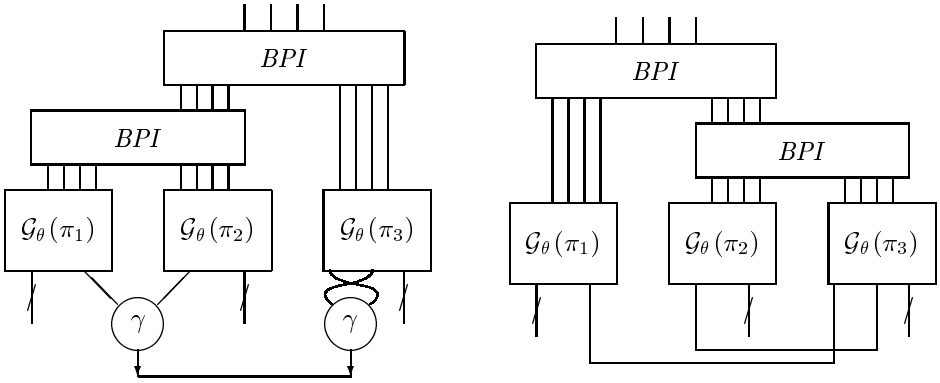
There are three cases to consider (note that the contraction and weakening cut-elimination steps are not possible as internal reductions—the corresponding nets are in normal form by Lemma 5.1).

If $\pi \Longrightarrow_w \pi'$ by an axiom cut-elimination step (AC), then $\mathcal{G}_\theta(\pi)$ and $\mathcal{G}_\theta(\pi')$ are given by the following two nets:



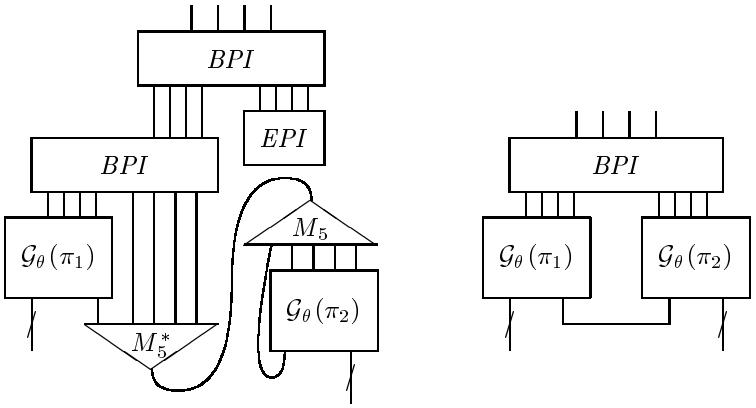
Now $\mathcal{G}_\theta(\pi) \equiv \mathcal{G}_\theta(\pi')$ since both nets have the same decomposition by Lemma 2.9. We note that the axiom cut-elimination step comes for free in this system of interaction nets—the concatenation of edges is just an edge—and thus no interactions are performed.

If $\pi \Longrightarrow_w \pi'$ by the multiplicative cut-elimination step $(\otimes, \parr)$, then $\mathcal{G}_\theta(\pi)$ and $\mathcal{G}_\theta(\pi')$ are given by the following two nets:



The result follows by using the $\gamma\gamma$ interaction rule, and then applying Lemma 2.9.

If $\pi \Longrightarrow_w \pi'$ by the dereliction cut-elimination step $(!, D)$, then $\mathcal{G}_\theta(\pi)$ and $\mathcal{G}_\theta(\pi')$ are given by the following two nets:



The result follows by the property of the multiplexing pair (M_5, M_5^*), and the use of Lemma 2.9 to eliminate the redundant part of the package interface. ■

Next we observe that the choice of the packing interface used in $\mathcal{G}_\theta(\pi)$ plays no role in the above proof: any net $N \equiv \mathcal{G}_\theta(\pi)$ reduces to a net $N' \equiv \mathcal{G}_\theta(\pi')$. We shall write $\Longrightarrow_\equiv$ for such a reduction step modulo $\equiv$, and both $\Longrightarrow_\equiv^*$ and $\Downarrow_\equiv$ have the obvious meanings.

We can now state the main result for package reduction.

THEOREM 5.1. *Let π be a proof inside an exponential box, then:*

$$\pi \Downarrow_w \pi' \iff \mathcal{G}_\theta(\pi) \Downarrow_\equiv \mathcal{G}_\theta(\pi')$$

Proof. $(\Rightarrow)$: By induction over the length of the reduction sequence using Lemma 5.3. The net $\mathcal{G}_\theta(\pi')$ is in normal form by Lemma 5.1. $(\Leftarrow)$ is a straightforward consequence of the confluence property of interaction nets (Lemma 2.1). ■

Using this result for internal reductions allows us to prove that weak cut-elimination steps can be simulated in full generality.

LEMMA 5.4. *If* $\pi \Longrightarrow_w \pi'$, *then* $\mathcal{G}(\pi) \Longrightarrow_{\equiv}^{*} \mathcal{G}(\pi')$.

Proof. The proof proceeds by cases on $\pi \Longrightarrow_w \pi'$. As before, we can draw the nets $\mathcal{G}(\pi)$ and $\mathcal{G}(\pi')$, and show that there is a sequence of interactions that performs the transformation. If the cut-elimination step is an internal reduction, then we use Lemma 5.3 (and thus we work again modulo $\equiv$). The remaining cases are outlined below:

- If $\pi \Longrightarrow_w \pi'$ by an axiom cut-elimination step, then $\mathcal{G}(\pi) = \mathcal{G}(\pi')$ (there are no interactions).
- If $\pi \Longrightarrow_w \pi'$ by a multiplicative cut-elimination step $(\otimes, \parr)$, then we apply the $\overset{\gamma\gamma}{\Longrightarrow}$ interaction rule to obtain the required net transformation.
- If $\pi \Longrightarrow_w \pi'$ by a weakening cut-elimination step, then by definition, the context of the promotion rule must be empty, and the proof being erased must be weak cut-free (closed cut-elimination step). The net M_5 used in the construction of the promotion rule can be erased, and then we apply the Erasing Lemma 2.6.
- If $\pi \Longrightarrow_w \pi'$ by a dereliction cut-elimination step, then using the fact that (M_5, M_5^*) form a multiplexing pair, we can just use Lemma 2.5.
- If $\pi \Longrightarrow_w \pi'$ by a contraction cut-elimination step, then by definition, the context on the promotion rule must be empty and the proof being copied must be weak cut-free (closed cut-elimination step). Using the fact that δ can duplicate the net M_5, we can apply the Duplication Lemma 2.7.

■

We can now put all the pieces together to obtain the main result of this paper.

THEOREM 5.2. *If* π *is a proof of* Γ, *where there are no occurrences of* $!$ *or* $?$ *in* Γ, *then*

$$\pi \Downarrow_w \pi' \iff \mathcal{G}(\pi) \Downarrow_{\equiv} \mathcal{G}(\pi')$$

Proof. First, by Theorem 4.1 observe that π' is a cut-free proof.

($\Rightarrow$): By induction over the reduction sequence, using Lemma 5.4. $\mathcal{G}(\pi')$ is a net in normal form by Lemma 5.1.

($\Leftarrow$): We show that if $\mathcal{G}(\pi) \Downarrow N$, then $\pi \Downarrow_w \pi'$ and $N = \mathcal{G}(\pi')$. By the cut-elimination theorem for linear logic there is a unique π' such that $\pi \Downarrow_w \pi'$, and by confluence of interaction nets $N \equiv \mathcal{G}(\pi')$. ■

Since in this case weak reduction coincides with cut-elimination, we obtain a system of interaction that computes cut-free proofs, where nets in normal form correspond to the translation of cut-free proofs. An important application of this system of interaction nets is for the encoding of functional programs, and in this case we are interested in evaluating programs at some base type. These cases are captured by the above theorem.

6. ADDITIVES

In this section we show how the additives of linear logic can be encoded using the interaction combinators. The additives are presented separately because the results are not quite as sharp as those for the multiplicatives and exponentials: only the cut-elimination steps

on the main conclusion of the $\&$ rule will be encoded. Consequently, the $\&$ commutation rule (see below) is not captured by the system. Such a constraint means that in general we will not be able to obtain cut-free proofs with the encoding, but has the advantage of not duplicating proofs unnecessarily. We begin by recalling the rules and cut-elimination steps for the additives, which consist of the $\&$ and $\oplus$ rules:

$$\frac{\Gamma, A \qquad \Gamma, B}{\Gamma, A \& B}\,(\&) \qquad \frac{A, \Gamma}{A \oplus B, \Gamma}\,(\oplus_l) \qquad \frac{B, \Gamma}{A \oplus B, \Gamma}\,(\oplus_r)$$

The distinguishing feature of the $\&$ rule is that the context Γ is shared by both premises. Under cut-elimination, only one of the premises will be selected, and the other will be erased. The two key cut-elimination steps for the additives are the following:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, A} \qquad \dfrac{\pi_2}{\Gamma, B}\,(\&)}{\Gamma, \boldsymbol{A \& B}} \qquad \dfrac{\dfrac{\pi_3}{A^\perp, \Delta}\,(\oplus_l)}{\boldsymbol{A^\perp \oplus B^\perp}, \Delta}}{\Gamma, \Delta}\,(\mathsf{Cut}) \quad \overset{(\&, \oplus_l)}{\Longrightarrow}_{ce} \quad \dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{A}} \qquad \dfrac{\pi_3}{\boldsymbol{A^\perp}, \Delta}}{\Gamma, \Delta}\,(\mathsf{Cut})$$

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, A} \qquad \dfrac{\pi_2}{\Gamma, B}\,(\&)}{\Gamma, \boldsymbol{A \& B}} \qquad \dfrac{\dfrac{\pi_3}{B^\perp, \Delta}\,(\oplus_r)}{\boldsymbol{A^\perp \oplus B^\perp}, \Delta}}{\Gamma, \Delta}\,(\mathsf{Cut}) \quad \overset{(\&, \oplus_r)}{\Longrightarrow}_{ce} \quad \dfrac{\dfrac{\pi_2}{\Gamma, \boldsymbol{B}} \qquad \dfrac{\pi_3}{\boldsymbol{B^\perp}, \Delta}}{\Gamma, \Delta}\,(\mathsf{Cut})$$

where $(\&, \oplus_l)$ (resp. $(\&, \oplus_r)$) erases the proof π_2 (resp. π_1). If the cut rule is not used on the principal formulas, then the commutation rules for $\oplus_l$ or $\oplus_r$ may be used. Both of these follow the same pattern, we just show the case for $\oplus_l$:

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \qquad \dfrac{\dfrac{\pi_2}{C, \Delta, A}}{\boldsymbol{C}, \Delta, A \oplus B}\,(\oplus_l)}{\Gamma, \Delta, A \oplus B}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \qquad \dfrac{\pi_2}{\boldsymbol{C}, \Delta, A}}{\Gamma, \Delta, A}\,(\mathsf{Cut}) \qquad \dfrac{\Gamma, \Delta, A}{\Gamma, \Delta, A \oplus B}\,(\oplus_l)$$

However, the following commutation rule for the $\&$ will *not* be permitted:

$$\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \qquad \dfrac{\dfrac{\pi_2}{C, \Delta, A} \qquad \dfrac{\pi_3}{C, \Delta, B}\,(\&)}{\boldsymbol{C}, \Delta, A \& B}}{\Gamma, \Delta, A \& B}\,(\mathsf{Cut}) \quad \rightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \qquad \dfrac{\pi_2}{\boldsymbol{C}, \Delta, A}}{\Gamma, \Delta, A}\,(\mathsf{Cut}) \qquad \dfrac{\dfrac{\pi_1}{\Gamma, \boldsymbol{C^\perp}} \qquad \dfrac{\pi_3}{\boldsymbol{C}, \Delta, B}}{\Gamma, \Delta, B}\,(\mathsf{Cut})}{\Gamma, \Delta, A \& B}\,(\&)$$

The motivation for this choice is that the proof π_1 has been duplicated, potentially unnecessarily. Any later cut on the main conclusion of this proof will cause one of the premises to be erased, including the proof π_1 which has just been duplicated. Of course, if no cut ever happens, then we fail to obtain cut-free proofs with this system. Nevertheless this seems to be the most natural encoding of the additives, at least from a computing perspective.

We next extend the notion of weak cut-elimination to include the additives. We add the cut-elimination steps given above, but we treat the $\&$ connective in the same way as the

promotion rule: only *internal* reductions will be allowed to take place above the rule. This choice is motivated from two perspectives: first, because one of the premises will be erased during cut-elimination we should postpone reduction, and secondly it provides the easiest interaction net encoding.

The following result, which is an extension of Theorem 4.1, states the conditions under which weak cut-elimination is sufficient to obtain cut-free proofs, i.e., the $\&$ commutation rule is not required to obtain these proofs.

THEOREM 6.1. *If π is a proof of Γ, where Γ does not contain any exponentials (? or !) or the $\&$ connective, then $\pi \Downarrow_w \pi'$, where π' is cut-free.*

Proof. The proof is a straightforward extension of the proof of Theorem 4.1. We just show the additional case required for the additives. Assume for a contradiction that π' is weak-cut-free, but not cut-free. Let h be the height of one of the lowest cuts in the proof (with respect to the conclusion). Since the proof is weak cut-free, the only additional case is for a cut on the context of the $\&$ rule:

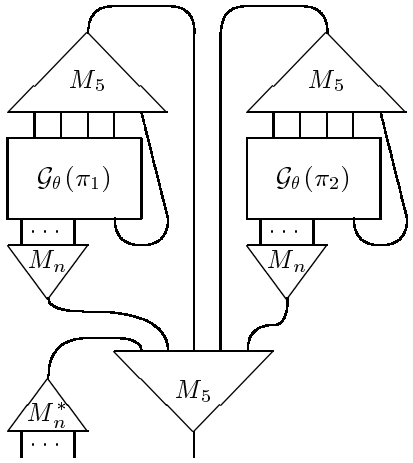
$$\dfrac{\Gamma, \boldsymbol{C} \qquad \dfrac{C^\perp, \Delta, A \qquad C^\perp, \Delta, B}{\boldsymbol{C^\perp}, \Delta, A\&B}\,(\&)}{\Gamma, \Delta, A\&B}\,(\mathsf{Cut})$$

Since there are no lower cuts in this branch of the proof of height $h' < h$, the formula $A\&B$ must appear in the conclusion of the proof, which is a contradiction. ■

The interesting aspect of the encoding of the $\&$ rule is that we must capture the idea of sharing the context Γ between the premises of the rule. There are two possible choices that we can take here. The first is to use δ agents, one for each conclusion in Γ. This solution permits the additive commutation rule, and indeed gives priority of this rule over cut-elimination on the main conclusion of the rule. As a consequence, many proofs will be duplicated, even though it may be known that only one of the premises of the $\&$ is required. To avoid this excessive duplication, and to put priority on the main formula of the rule, we use an alternative method which respects the fact that additives *share* rather than duplicate the context.

The encoding of the logical rules is given by the following three cases.

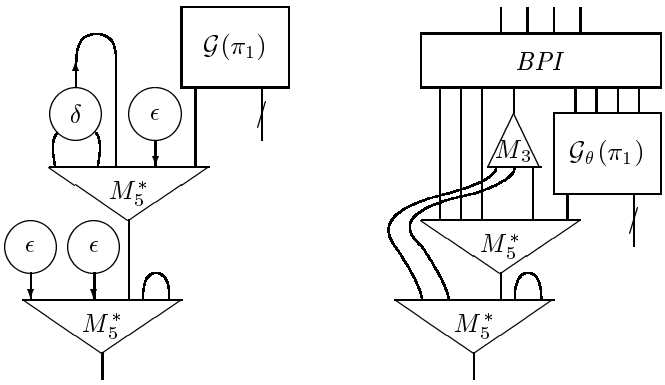
• Let π_1 be a proof of Γ, A and π_2 be a proof of Γ, B, then we can build a proof π of $\Gamma, A\&B$ using the $\&$ rule. $\mathcal{G}(\pi)$ is then defined as the following net, where n is the size of the context Γ (i.e. $\Gamma = A_1, \ldots, A_n$).



For each premise, we build packages $\mathcal{G}_\theta(\pi_1)$ and $\mathcal{G}_\theta(\pi_2)$ in exactly the same way as for the promotion rule. The M_5 nets at the top form a multiplexing pair with the net M_5^* used in the encoding of the $\oplus$ rules below. Next, the n free edges representing the context Γ are multiplexed using M_n nets for the encoding of π_1 and π_2, and the context for the net representing π is then given by the free edges of the net M_n^*, where (M_n, M_n^*) is a multiplexing pair. The free edges corresponding to the conclusions Γ, A from π_1, and Γ, B from π_2 are then grouped together with the edge representing the context Γ of the rule using an M_5 net. The principal port of this M_5 net corresponds to the conclusion $A\&B$. Thus we have a principal port representing the main conclusion, and auxiliary ports for the context: no interactions will be possible on the context Γ. The encoding of $\mathcal{G}_\theta(\pi)$ is exactly the same structure, where for $n > 0$ we simply add the EPI net to build the correct interface, or if $n = 0$ then we must also extract the additional ϵ agents used in the construction of M_0 and M_0^*.

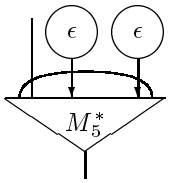
The idea of this encoding is that when an interaction takes place with the main conclusion of the proof, the context will be connected to the net representing either π_1 or π_2, and the other erased. This will be done by the nets representing the $\oplus$ rules, which are covered in the following two cases.

- Let π_1 be a proof of A, Γ, then we can build a proof π of $A \oplus B, \Gamma$ using the $\oplus_l$ rule. $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are then given by the following nets:



where both M_5^* nets form a multiplexing pair with the M_5 nets used in the encoding of the $\&$ rule. One can see from this construction that if connected to the encoding of the $\&$ rule, the net $\mathcal{G}_\theta(\pi_2)$ will be erased, and the net $\mathcal{G}_\theta(\pi_1)$ will be unpacked to give $\mathcal{G}(\pi)$ as required.

- Let π_1 be a proof of B, Γ, then we can build a proof π of $A \oplus B, \Gamma$ using the $\oplus_r$ rule. $\mathcal{G}(\pi)$ and $\mathcal{G}_\theta(\pi)$ are almost identical to the $\oplus_l$ rule above, where we replace the order of the connections on the bottom M_5^* net to be the following:



The net $\mathcal{G}_\theta(\pi)$ is then obtained by erasing the two ϵ agents and connecting the free edges to the M_3 net as in the previous case.

This completes the translation of the additives into interaction combinators. We next need to extend some of the results of Section 5 to cover the additive fragment of linear logic. First, Lemmas 5.1 and 5.2 can be extended without any difficulty. We can then give the two main results for additive reduction:

LEMMA 6.1 (Additive Reduction). *If $\pi \Longrightarrow \pi'$ by an additive cut-elimination step, then:*

1.Package reduction: $\mathcal{G}_\theta(\pi) \Longrightarrow_{\equiv}^* \mathcal{G}_\theta(\pi')$.
2.External Reduction: $\mathcal{G}(\pi) \Longrightarrow_{\equiv}^* \mathcal{G}(\pi')$.

Proof. Both follow the same pattern as for the multiplicative and exponential parts, where we can draw the corresponding nets and show that there is a reduction sequence between them, modulo $\equiv$. The Unpacking Lemma 2.5 is required to show that the packaged proof can be recovered, and the Erasing Lemma 2.6 is used to show that the unused premise can be completely erased. The only difficulty in the proof is to show that for package reduction both nets have the same decomposition for the multiplexing nets collecting the extracted ϵ agents. However, this can be proved by showing that both nets unpack to the same net, and the result follows by the Erasing Lemma 2.6. ■

Finally we conclude this section with a general statement about the encoding of linear logic into the interaction combinators, which is a direct consequence of the above results.

THEOREM 6.2. *If π is a proof of Γ, where Γ does not contain any occurrences of $!$, $?$, or $\&$, then:*

$$\pi \Downarrow_w \pi' \iff \mathcal{G}(\pi) \Downarrow_{\equiv} \mathcal{G}(\pi')$$

where π' is a cut-free proof.

7. λ-CALCULUS

There are several well-known translations of the λ-calculus into linear logic proofs (both for typed and untyped λ-calculi), see for instance [5]. Using these translations this paper directly offers (by composition) interaction net encodings for the λ-calculus. With the addition of constants in the calculus and in the system of interaction nets, one can thus obtain an implementation of a minimalistic functional programming language, such as PCF [18].

We give here two translations of the λ-calculus into interaction combinators, and briefly mention the properties of the resulting systems. We begin with the so-called "call-by-value" translation, which is based on translating A as $!A$ and $A \Rightarrow B$ as $!(A^{\perp} \parr B)$. Briefly, this means that:

- the translation of an abstraction $\lambda x.t$ requires the use of the $\parr$ and promotion rules (with the use of weakening if the variable x does not occur in the free variables of t);
- application requires the use of $\otimes$ and dereliction rules (with the use of contraction if there are free variables common to both t and u);
- a variable is translated as an instance of the axiom.

The second translation we give, the so-called "call-by-name" translation, is based on translating A as A, and $A \Rightarrow B$ as $?A^{\perp} \parr B$. Briefly, this means that:

- the translation of variables becomes a dereliction;
- abstraction requires the use of $\parr$;
- application uses $\otimes$ and promotion for the argument.

These brief comments will become clearer when we give the actual translations below.

To simplify the compilations of the λ-calculus into interaction combinators, it is useful to extend the syntax of the λ-calculus with explicit discarding and copying constructs, which are written as $E_x(t)$ and $C_x^{y,z}(t)$ respectively. The first says that x does not occur in t, and the second says that if x occurs twice in t then we rename the two occurrences of x by y and z, which are combined by the copying construct. If x occurs more than twice then we can use the second rule repeatedly so all occurrences of the variable x get a unique name. Additionally, to monitor the progress of substitutions it is useful to include the notion explicitly: we use the usual notation $t[u/x]$. Note that there are trivial encodings of the λ-calculus into this extension which do nothing more than variable counting. Two examples of this notation that we use later in this paper are the combinators $\mathbf{K} = \lambda xy.E_y(x)$ and $\mathbf{S} = \lambda xyz.C_z^{u,v}(xu)(yv)$. Using this notation, all variables occur exactly once in the body of a λ-term.

There is an obvious notion of *free variables* for this enriched λ-calculus, which we represent as an ordered sequence of variables, written as $[x; y; z]$, etc., defined in the following way:

$$\begin{array}{lcl}
\mathsf{fv}(x) & = & [x] \\
\mathsf{fv}(\lambda x.t) & = & \mathsf{fv}(t) - [x] \\
\mathsf{fv}(tu) & = & \mathsf{fv}(t) + \mathsf{fv}(u) \\
\mathsf{fv}(t[u/x]) & = & (\mathsf{fv}(t) - [x]) + \mathsf{fv}(u) \\
\mathsf{fv}(E_x(t)) & = & \mathsf{fv}(t) + [x] \\
\mathsf{fv}(C_x^{y,z}(t)) & = & (\mathsf{fv}(t) + [x]) - [y; z]
\end{array}$$

where $+$ is the concatenation, and $-$ is the removal of the first occurrence of the element. We will also use the notation $\vec{x}$ for an ordered sequence of free variables.

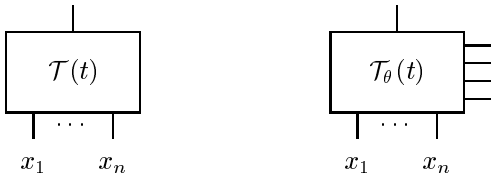
The reduction system for this calculus is then given by the following set of rules:

$$
\begin{array}{lll}
(\lambda x.t)u & \to\ t[u/x] & \\
x[v/x] & \to\ v & \\
(tu)[v/x] & \to\ (t[v/x])u & (x \in \mathsf{fv}(t)) \\
(tu)[v/x] & \to\ t(u[v/x]) & (x \in \mathsf{fv}(u)) \\
(E_y(t))[v/x] & \to\ E_y(t[v/x]) & \\
(E_x(t))[v/x] & \to\ E_{\vec{x}}(t) & (\vec{x} = \mathsf{fv}(v)) \\
(C_y^{z,z'}(t))[v/x] & \to\ C_y^{z,z'}(t[v/x]) & \\
(C_x^{y,z}(t))[v/x] & \to\ C_{\vec{x}}^{\vec{y},\vec{z}}(t[v[\vec{y}/\vec{x}]/y])[v[\vec{z}/\vec{x}]/z] & (\vec{y},\vec{z}\ \text{fresh}, \vec{x} = \mathsf{fv}(v)) \\
(\lambda y.t)[v/x] & \to\ \lambda y.t[v/x] &
\end{array}
$$

where the notation $E_{\vec{x}}(t)$ is used as an abbreviation for $E_{x_1}(E_{x_2}(\cdots E_{x_n}(t)\cdots))$, and similarly for $C_{\vec{x}}^{\vec{y},\vec{z}}(t)$. These rules are nothing other than the usual rules for β-reduction, where the meta-operation of substitution becomes a rewrite rule, and we have added a few rules which give the rewrite semantics of the additional constructs. Note that we assume the variable convention (bound names are always chosen to be different [2]), which is convenient and also is reflected in the interaction system (variable names play no role).

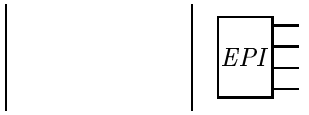
7.1. The "call-by-value" $!(A \multimap B)$ Translation

As with the translation of linear logic proofs, we define two translations of the λ-calculus into interaction nets. The first one $\mathcal{T}(t)$ which is the main translation, and the packaged net $\Theta(\mathcal{T}(t))$ which we shall write as $\mathcal{T}_\theta(t)$. For a term t in the λ-calculus, with $\mathsf{fv}(t) = [x_1; \ldots; x_n]$, the general forms of these translations are given below:



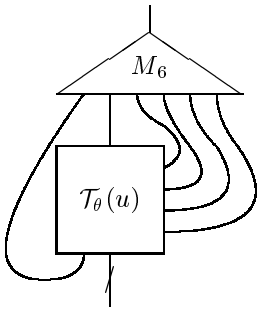
The edge at the top corresponds to the root of the term, and the edges $x_1, \ldots, x_n$ correspond to the free variables of the term t. The four additional edges used in the net $\mathcal{T}_\theta(t)$ correspond as before to the free edges created by extracting the δ and ϵ agents: from top to bottom we have the collection of l, r and p edges for the δ and the collection of ϵ edges. We will drop the labeling of the free variables since this is derived directly from the term, and the order is preserved, and we shall also use the bus notation for multiple edges as before.

Variable. If t is a variable, say x, then $\mathcal{T}(t)$ and $\mathcal{T}_\theta(t)$ are given respectively as:



where we have used again the net EPI to provide the correct interface for the packaging function when there are no occurrences of δ and ϵ in the net.

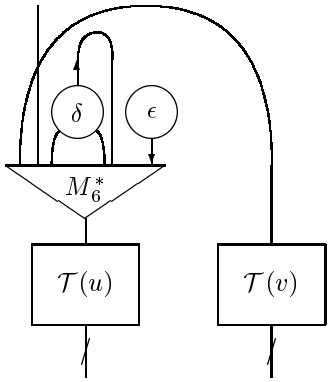
Abstraction. If t is an abstraction $\lambda x.u$, then $\mathcal{T}(t)$ is given by the following net:



where we have assumed without loss of generality that the free variable x occurs in the leftmost position of the free variables of $\mathcal{T}_\theta(u)$. The net M_6, where (M_6, M_6^*) is a multiplexing pair, groups together the bound variable x, the root of the term, and the free edges from the extracted δ and ϵ agents respectively. Since $\mathcal{T}(t)$ is free from both δ and ϵ agents, the net for $\mathcal{T}_\theta(t)$ (not shown) is the same net with the EPI net used to provide the correct interface.

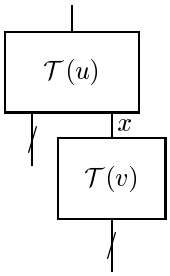
We remark that for the translation that we are using, this corresponds to the use of the promotion and $\parr$ rules: thus the net M_6 is the M_5 net for the promotion together with a γ agent for the $\parr$.

Application. If t is an application uv, then $\mathcal{T}(t)$ is given by the following net:



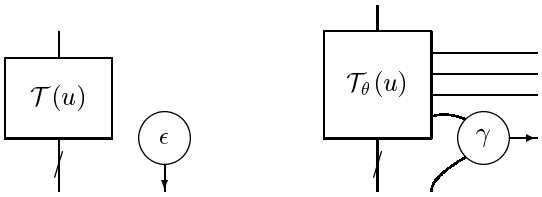
The net M_6^*, which forms a multiplexing pair with the M_6 net used in the abstraction above, groups together (from left to right) the argument, the result (root), and the code for opening a package. Thus this corresponds to the encoding of a dereliction and a $\otimes$ rule. The net M_6^* is thus the net M_5^* together with a γ agent representing the tensor. The net for $\mathcal{T}_\theta(t)$ (not shown) is constructed by erasing the δ and ϵ agents in the above diagram, and merging all the free edges together using two BPI nets.

Substitution. If t is $u[v/x]$, then $\mathcal{T}(t)$ is given by the following net:

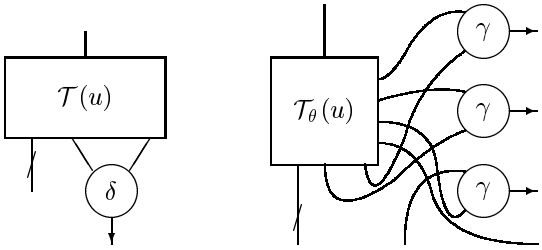


where we have assumed without loss of generality that the free variable x occurs in the rightmost position of the net $\mathcal{T}(u)$. The net $\mathcal{T}_\theta(t)$ (not shown) is then constructed by merging the corresponding edges with a BPI net as usual.

Erase. If t is $E_x(u)$, then $\mathcal{T}(t)$ and $\mathcal{T}_\theta(t)$ are given by the following nets, where we use the ϵ agent:



Copy. If t is $C_x^{y,z}(u)$, then $\mathcal{T}(t)$ and $\mathcal{T}_\theta(t)$ are given by the following nets, where we use the agent δ to group together y and z into a single edge x:

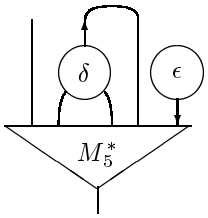


Note that we have assumed that the occurrences of y and z were the rightmost edges of $\mathcal{T}(u)$. Edges can always be swapped to put them in the correct position, which is analogous to the encoding of the exchange rule for linear logic. This completes the call-by-value compilation of the λ-calculus into interaction combinators.

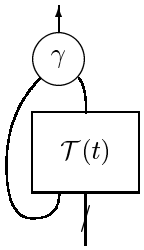
7.2. The "call-by-name" $!A \multimap B$ Translation

In this section we give an alternative encoding of the λ-calculus by using the so-called "call-by-name" (or $!A \multimap B$) translation of the λ-calculus into linear logic. As in the previous section we give an inductive translation over the structure of our λ-terms. The translation of Substitution, Erase and Copy are identical to the "call-by-value" translation, and thus we will not repeat those cases here. The three remaining cases are the following (we show only the $\mathcal{T}(t)$ translation):

Variable. If t is the variable x, then $\mathcal{T}(t)$ is encoded as a dereliction for linear logic:

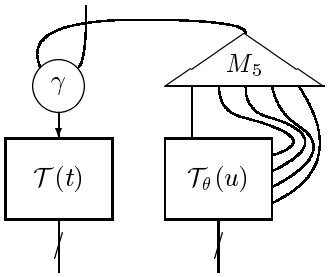


Abstraction. The encoding of $\lambda x.t$ is given by translating the body of the abstraction, and connecting the edge at the top of the net to the edge corresponding to the variable x, with a γ agent:



We have assumed in the diagram that x occurs in the leftmost position.

Application. The encoding of tu requires the use of promotion for the argument u, for which we build a package, and a γ agent to represent the application:



7.3. Comparing the Translations

To complete this study of encoding the λ-calculus using interaction combinators, we outline the essential differences between the two translations given, and also compare these to other interaction net encodings of the λ-calculus. We begin with some general remarks on the two translations given previously.

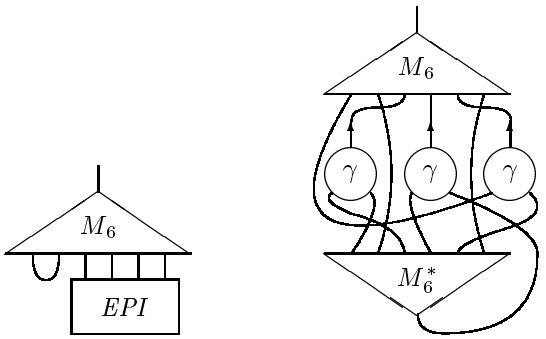
The essential difference between the call-by-name and the call-by-value translations is that the exponential box is used either to encode the argument or the function. This has two consequences:

- In the call-by-name translation the argument to an application is a package, which means that no internal duplication or erasing is possible. For the call-by-value translation it is the function which is packaged, thus erasing and copying are not possible under an abstraction.
- Apart from the fact that the positions of the exponentials are changed for the two translations, there is one other important difference in that with the CBN translation we do

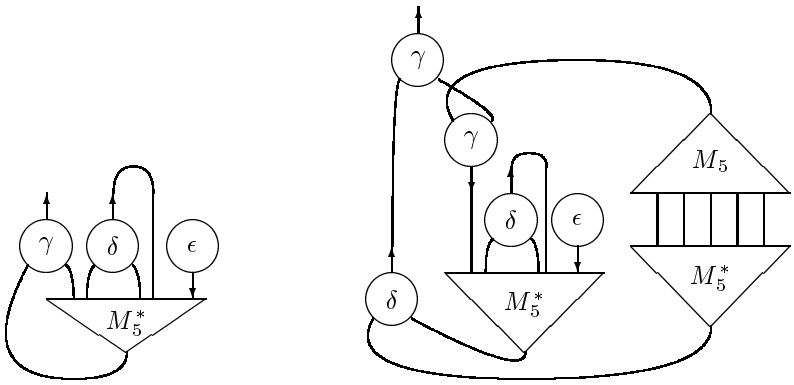
not have the possibility to group the M_5 nets for the packing with the γ agents used for the encoding of the application and abstraction. Consequently the number of interactions will be greater for the call-by-name translation.

To demonstrate the differences, we give some examples of the translations.

EXAMPLE 7.1. The following diagram shows the call-by-value translation of the example terms: $\lambda x.x$ and $\lambda x.xx$. We have simplified the package construction for the translation of $\lambda x.xx$ to keep the diagram simple.



The next diagram shows exactly the same terms ($\lambda x.x$ and $\lambda x.xx$) translated using the call-by-name translation. We remark that some simplification can also be done with this translation: the M_5 and M_5^* nets connected together on all the auxiliary ports can be simplified into a single edge.



The correctness of both these encodings of the λ-calculus is derived directly from the results obtained for linear logic. However, it is worth stating the consequences of some of the previous results in the context of the λ-calculus.

1. Lemma 5.2 (which states that this system of interaction nets represents proofs modulo the permutation rules) together with the fact that axiom cut-elimination steps are obtained for free gives many of the reduction rules as equivalences. In particular, for the call-by-value translation: If $t \to u$ by:

- $x[v/x] \to v$
- $(tu)[v/x] \to (t[v/x])u$ if $x \in \mathsf{fv}(t)$

- $(tu)[v/x] \to t(u[v/x])$ if $x \in \mathsf{fv}(u)$
- $(E_y(t))[v/x] \to E_y t[v/x]$
- $(C_y^{z,z'}(t))[v/x] \to C_y^{z,z'} t[v/x]$
- $(\lambda y.t)[v/x] \to \lambda y.t[v/x]$

then $\mathcal{T}(t) \equiv \mathcal{T}(u)$ and $\mathcal{T}_\theta(t) \equiv \mathcal{T}_\theta(u)$. Moreover, the net representation factors out the order of substitutions: $(t[v/x])[w/y]$ gives the same net as $(t[w/y])[v/x]$. Thus many reduction steps in the rewriting system are captured for free in this system of interaction nets. In particular *substitution through an abstraction is obtained for free*, which is generally regarded as being the most difficult (and computationally expensive) operation to implement in the λ-calculus. Similar comments apply to the call-by-name translation, where substitution to the head variable is obtained for free.

2. Lemma 5.3 (Package reduction) states which reductions can take place inside a package. For the call-by-value translation, this implies that duplication and erasing reductions are not permitted under an abstraction (all other reductions are allowed). For the call-by-name translation, the same condition is applied to the arguments.

3. Lemma 5.4 (Simulation of weak cut-elimination) gives the conditions when a λ-term can be reduced to normal form with this system of interaction nets. Specifically, the following reductions are permitted

- $(\lambda x.t)u \to t[u/x]$
- $(E_x(t))[v/x] \to t$ if $\mathsf{fv}(v) = \emptyset$
- $(C_x^{y,z}(t))[v/x] \to (t[v/y])[v/z]$ if $\mathsf{fv}(v) = \emptyset$

where the last two rules are external reductions which require that v is a weak normal form. Specifically, this shows the restrictions of only erasing and duplicating substitutions when they are closed. Properties of calculi where reductions can only be performed when the substitution is closed have also been investigated in [3].

One can observe from these translations that the nets M_6 and M_6^* for call-by-value (resp. M_5 and M_5^* for call-by-name) occur frequently, and it makes sense, with respect to efficiency, to include them in the system of combinators as new agents (the interaction rules with the other agents are the expected ones, easily derived from the definition of M_n and M_n^*). Moreover, recall that M_6 (resp. M_5), δ and ϵ also provide a complete system of interaction, since γ can be simulated in terms of M_6 (resp. M_5) and ϵ.

Both resulting systems have been implemented using an interaction net evaluator [16], and compared with the other systems of interaction mentioned in the introduction. We use λ-terms representing Church numerals, which provide an excellent set of test data and generate vast computations since application is exponentiation; they have also become the standard for testing the sharing ability of a reduction system. The basis for the comparison of the different systems is the total number of interaction steps required to reduce a net to normal form.

In the following table the columns represent 4 different evaluators under test, which we have implemented in a common framework. GAL is the optimal one, as reported in [6], ABR is reported in [13], YALE is reported in [14], and finally, **IC**(CBV) and **IC**(CBN) are the ones reported in this paper. We show the λ-term under test, and the numbers indicate the total number of interactions performed by each of the evaluators. The numbers

in parentheses indicate β-reductions, which corresponds to the number of interactions between the coding of an application and an abstraction. The number of β-reductions for GAL is thus the minimum that we can hope to get.

Term	GAL	ABR	YALE	**IC**(CBV)	**IC**(CBN)
22II	204(9)	56(11)	38(9)	66(9)	116(11)
222II	789(16)	304(42)	127(20)	278(18)	819(48)
33II	649(15)	332(45)	87(15)	322(15)	811(45)
322II	7055(21)	4457(531)	383(51)	3268(29)	14531(723)
223II	1750(19)	1046(132)	213(31)	869(22)	2860(148)

These comparisons are intended merely as a curiosity: can such a simple system of interaction nets for the λ-calculus perform well in comparison with other encodings, where the latter have been developed on the grounds of efficiency. The most remarkable result that we can see from this table is the level of sharing obtained (the number of β-reductions performed) for **IC**(CBV): the amount of sharing obtained is better than all the other evaluators, with the exception of the optimal one. Specifically, it offers more sharing than any of the other extant finite systems of interaction nets, which is clearly a positive sign for this evaluator. Some remarks with respect to the amount of sharing:

- As mentioned before, exponential commutative cuts are obtained for free in the call-by-value translation, and this implies that substitution through an abstraction is obtained without any reduction steps, and moreover any term which is copied will have this reduction "shared".
- For the call-by-name translation less sharing is obtained since copying and erasing reductions in the arguments are blocked, and thus potentially duplicated.

However, looking at the total number of interactions, **IC** is quite far from having the least number. We remark that:

- There are more interactions overall because of the additional interactions necessary to deal with the packing nets (which are multiplexing nets). Interaction combinators are more decomposed than other systems of interaction nets, and many interactions may be required to do a given rewiring which could be done in a single interaction in a different system. This explains the negative aspect of the experimental results given above.
- The call-by-value translation performs fewer interactions than the interaction net implementation of Lamping's algorithm, which indicates that there is less overhead using packages than indexes on the agents. However, the number of reductions is higher than other implementations of optimal reduction, for instance BOHM [1], which remains the reference for all new attempts at implementing the λ-calculus with interaction nets.

On a positive note, we can also observe that the complexity of each rewrite rule for the combinators is simpler than for the other systems, and one could hope for the development of a dedicated evaluator for the combinators which may be more efficient than a general purpose evaluator. Many optimizations remain to be investigated for this evaluator, for which there appears to be a lot of scope.

Finally we remark that this system of interaction may be better adapted towards a parallel implementation of the λ-calculus, since the number of interactions that can take place at any one time appears to be higher than for any extant system for the λ-calculus.

8. CONCLUSIONS

In this paper we have shown that the interaction combinators can be used to give an encoding of cut-elimination in linear logic and β-reduction in the λ-calculus. We believe that this is the simplest of all the known interaction nets encodings, and moreover experimental results indicate that the level of sharing of β-reduction interactions is better than for any other finite system of interaction nets. Additionally this system of interaction nets offers a representation of proofs in linear logic where the commutation rule for the exponentials comes for free, which, in this sense, improves upon proof nets as a syntax for linear logic free of commutation rules.

REFERENCES

1. A. Asperti, C. Giovannetti, and A. Naletto. The bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810, November 1996.
2. H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, second, revised edition, 1984.
3. M. Fernández and I. Mackie. Closed reduction in the λ-calculus. In J. Flum and M. Rodríguez-Artalejo, editors, *Proceedings of Computer Science Logic (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 220–234. Springer-Verlag, September 1999.
4. S. J. Gay. Combinators for interaction nets. In C. Hankin, I. Mackie, and R. Nagarajan, editors, *Theory and Formal Methods of Computing 94*, pages 63–84. Imperial College Press, September 1995.
5. J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
6. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26. ACM Press, January 1992.
7. G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *Proceedings of the 7th IEEE Symposium on Logic in Computer Science (LICS'92)*, pages 223–234. IEEE Press, 1992.
8. Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, January 1990.
9. Y. Lafont. From proof nets to interaction nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, number 222 in London Mathematical Society Lecture Note Series, pages 225–247. Cambridge University Press, 1995.
10. Y. Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997.
11. J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 16–30. ACM Press, January 1990.
12. J.-J. Lévy. *Réductions Correctes et Optimales dans le Lambda-Calcul*. Thèse d'état, Université Paris VII, January 1978.
13. I. Mackie. *The Geometry of Implementation*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, September 1994.
14. I. Mackie. YALE: Yet another lambda evaluator based on interaction nets. In *Proceedings of the 3rd International Conference on Functional Programming (ICFP'98)*, pages 117–128. ACM Press, 1998.
15. I. Mackie. Interaction nets for linear logic. *Theoretical Computer Science*, 247(1):83–140, September 2000.
16. J. S. Pinto. Sequential and concurrent abstract machines for interaction nets. In J. Tiuryn, editor, *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1784 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 2000.
17. J. S. Pinto. *Parallel Implementation with Linear Logic*. PhD thesis, École Polytechnique, February 2001.
18. G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–256, 1977.